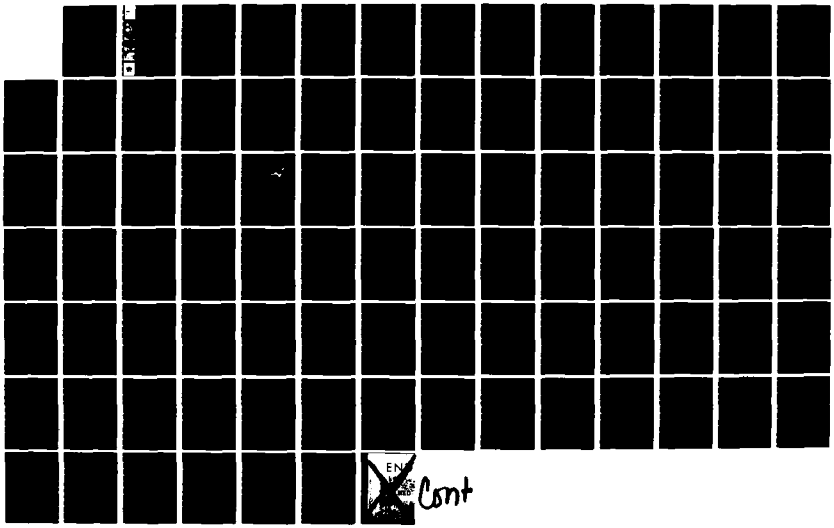
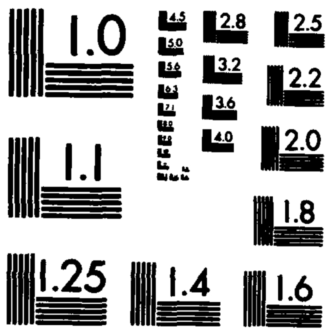
AD-A134 141 ACCURACY CONSIDERATIONS WHEN USING SOME MINICOMPUTERS 1/2
FOR SCIENTIFIC AND..(U) ARMY ENGINEER WATERWAYS
EXPERIMENT STATION VICKSBURG MS W F INGRAM ET AL.
UNCLASSIFIED SEP 83 WES/TR/K-83-2 F/G 12/1 NL

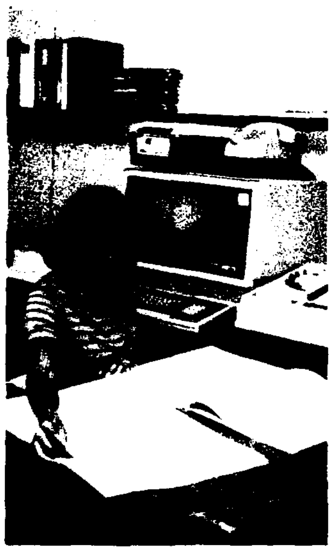

Cont


1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

(2)

US Army Corps of Engineers

AD-A134 141



AUTOMATIC DATA PROCESSING CENTER

DTIC FILE COPY

TECHNICAL REPORT K-83-2

ACCURACY CONSIDERATIONS WHEN USING SOME MINICOMPUTERS FOR SCIENTIFIC AND ENGINEERING PROBLEMS

by

Windell F. Ingram

University of Southwestern Louisiana
Lafayette, La. 70501

and

Deborah F. Dent, N. Radhakrishnan

Automatic Data Processing Center
U. S. Army Engineer Waterways Experiment Station
P. O. Box 631, Vicksburg, Miss. 39180

September 1983
Final Report

Prepared for Office, Chief of Engineers, U. S. Army
Washington, D. C. 20314

83 10 2. 0 5

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Report K-83-2	2. GOVT ACCESSION NO. AD-A134 141	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ACCURACY CONSIDERATIONS WHEN USING SOME MINICOMPUTERS FOR SCIENTIFIC AND ENGINEERING PROBLEMS		5. TYPE OF REPORT & PERIOD COVERED Final report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Windell F. Ingram Deborah F. Dent N. Radhakrishnan		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Southwestern Louisiana, Lafayette, La. 70501 & U. S. Army Engineer Waterways Experiment Station, ADP Center, Vicksburg, Miss. 39180		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office, Chief of Engineers, U. S. Army Washington, D. C. 20314		12. REPORT DATE September 1983
		13. NUMBER OF PAGES 78
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Available from National Technical Information Service, 5285 Port Royal Road, Springfield, Va. 22151.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Arithmetic
Computer applications
Mathematical programming
Microcomputers
Minicomputers

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report presents some fundamentals of floating-point (FP) arithmetic and some causes of substantial loss of accuracy and examines the internal FP representations for large-scale minicomputers. Results of tests of relative accuracy, which involved executing a set of test programs on several representative systems, are given along with some conclusions regarding the accuracy inherent in each system. The systems used were the Harris Series 500, VAX 11/780, Prime 550, and IBM 4331. The CDC CYBER 6600 was used to establish a baseline for comparing the results.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

PREFACE

This report examines the accuracy of some minicomputers in performing floating-point computations. The work was accomplished with funds provided to the Automatic Data Processing (ADP) Center, U. S. Army Engineer Waterways Experiment Station (WES), Vicksburg, Miss., by the Coordination and Integration Branch (formerly the CE-80 Project), Information Resource Management Division, Resource Management Directorate, Office, Chief of Engineers (OCE), Washington, D. C. The need for this work was identified by the CE-80 Software Working Group consisting of Dr. N. Radhakrishnan, WES, Chairman; Ms. Shirley Hendry, North Pacific Division; Messrs. Bob Williams, Kansas City District; Leonard (Skip) Manson, New Orleans District; Jack Pickett, OCE; Earl V. Edris, WES; and Wassil J. Lagoey, Engineer Automation Support Activity. The Group worked under the direction of Mr. Harry Hardin, Assistant Project Manager, CE-80 Project. Mr. Dick Colver was the CE-80 Project Manager.

The report was written by Mr. Windell F. Ingram, formerly with the ADP Center, WES, and now with the University of Southwestern Louisiana; Mrs. Deborah F. Dent, Research and Development Software Group, ADP Center, WES; and Dr. Radhakrishnan, Special Technical Assistant, ADP Center, WES.

The authors would like to thank Dr. Leroy Emkin, Professor of Civil Engineering, Georgia Institute of Technology, and Mr. Stacey Stringer, System Programmer, for their permission to use and their assistance in using the VAX 780 and Mr. Samuel McCutchen, Director of Management Information System, U. S. Army Mobility Equipment Research and Development Command, Fort Belvoir, Va., and Mr. Dave Morgan, System Programmer, for their permission to use and their assistance in using the Prime 550.

The work reported herein was under the direction of Dr. Radhakrishnan; Mr. Harry F. Hardin was the OCE point of contact.

Commander and Director of WES during the work was COL Tilford C. Creel, CE. Technical Director was Mr. F. R. Brown.

CONTENTS

CONVERSION FACTORS, NON-SI TO SI (METRIC) UNITS OF MEASUREMENT

Non-SI units of measurement used in this report can be converted to SI (metric) units as follows:

Multiply	By	To Obtain
feet	0.3048	meters
inches	0.0254	meters
inch-pounds (mass)	0.01152125	kilogram-meters
kips (1000 lb mass)	453.59237	kilograms
pound (mass)-square inches	0.00029264	kilogram-square meters
pounds (force) per square foot	47.88026	pascals
pounds (force) per square inch	6.894757	kilopascals
pounds (mass)	0.45359237	kilograms
pounds (mass) per cubic foot	16.01846	kilograms per cubic meter
pounds (mass) per cubic inch	27,679.905	kilograms per cubic meter
pounds (mass) per inch	17.85797	kilograms per meter

ACCURACY CONSIDERATIONS WHEN USING SOME MINICOMPUTERS FOR SCIENTIFIC AND ENGINEERING PROBLEMS

PART I: INTRODUCTION

Background

1. Floating-point (FP) computations are routinely used in programming for scientific and engineering (S&E) applications. Novice computer users tend to implicitly trust the computer to produce correct answers and might not question the accuracy of results to as many significant digits as might be printed. Experienced users tend to become disillusioned after experiences with erroneous answers and eventually wary of FP computations that produce believable answers that are nevertheless wrong. Indeed, FP computation is inherently inexact and can easily be inadvertently misused. Experience in the Corps of Engineers has shown that FP processing on some computers is too imprecise for many common S&E applications.

2. One approach to dealing with this problem is to require that all real variables be double-precision, but this exacts penalties in main memory required and often in execution time. The penalties frequently are so severe that this approach is impractical for many applications. Therefore, the tendency is to process precision-sensitive applications on long-word-length machines; e.g., large-scale CDC systems with 60-bit words. In some cases, careful analysis of a program might indicate that acceptable accuracy is attainable with shorter word lengths if minor program modifications are made; e.g., double-precision calculations only in a few critical areas. While the modifications may be minor, the analysis often is not and might involve expenditure of substantial human and machine resources. Such analyses are rarely pursued by the engineer programmer since his expertise is usually not in numerical analysis. From his perspective, the easiest and most cost-effective solution has often been simply to move to a long-word-length machine. This not only eliminates many actual problems caused by inadequate computational accuracy, but also allays fears that accuracy problems are lurking in every program. The engineer programmer then has confidence in the machine and can concentrate on algorithms for his application rather than on computational error analysis.

3. Within recent years, computer systems classed as large-scale minis or superminis have become available, offering much of the sophistication and power of mainframes at a much lower cost. These machines are exemplified by the three families addressed herein: the VAX 11, the PRIME 550, and the Harris Series 500. The functional capabilities and performance/price ratios of such systems make them very attractive to many engineers. Some envision such machines dedicated to a relatively small group of engineers--perhaps a group, section, or branch--where there is no contention for machine resources from management and business programs or from other users outside the group. The ability to manage and control one's own machine resources, rather than sharing a central facility, has considerable appeal to many. Whether used as a "private" system or a central facility to provide a part of the computational requirements of a larger Corps organization (e.g., an Engineer District), this class of computer system holds great interest within the Corps of Engineers. Witness the recent procurement of Harris 500 systems and the planned Corps of Engineers' Automation Plan (CEAP) for local processor systems.

Objectives

4. The objectives of this report are to present some fundamentals of FP arithmetic and some causes of substantial loss of accuracy, to examine the internal representations for each of the three machine families, to test the relative accuracy of the machine families by executing a set of test programs on representative systems, and to draw some conclusions regarding the accuracy inherent in each machine family. The test programs were chosen to include both "textbook" and "real world" problems. The problems consisted of a simple arithmetic problem, two numerical analysis problems, and three S&E problems. The same set of data for each program was used on each of the systems. The data for the S&E programs represented normal conditions and were not "cooked-up" to show word-length problems. The minicomputers used in this study were the Harris 500, VAX 11/780, and PRIME 550. Runs were also made on an IBM 4331, to permit comparison with the very well known and widely available IBM FP representation, and on the CDC CYBER 6600, a large-scale system whose results were used as a base for comparing the other results.

Scope

5. It is not intended to present details of each machine's internal algroithms for performing FP operations (i.e., step-by-step register moves, shifts, adds, etc.), nor to identify anomalies that may be present in each vendor's implementation of the operations, since such information is not readily available and is not necessary for drawing conclusions regarding relative accuracies for the three families in question. All runs were also made on a CDC CYBER system to serve as a baseline. The systems were not selected due to their competitiveness in either performance or price. Rather, they were selected simply as three available systems with FP architectures representative of the three families of systems.

Efficiency Considerations

6. FP operations are implemented on most modern minicomputer systems as part of the computer architecture--either through hardware FP units or through microprogramming. How the FP operations are implemented greatly affects machine performance for S&E applications; i.e., hardware implementations are generally much faster than microprogrammed implementations. While only a very cursory and inconclusive comparison of program execution times is made herin, it should be noted that a well-planned, rigorous comparison of FP computation speeds is an important factor in selecting a minicomputer for S&E applications. Also note that all three families of systems have hardware implementations either as standard equipment or as options.

PART II: FUNDAMENTALS OF FLOATING-POINT ARITHMETIC

Basic Notation

7. A machine's representation of FP numbers is the computer equivalent of the human representation used to express very large or very small numbers; i.e., the familiar scientific notation. The number consists of two parts: a signed part usually called the fraction or mantissa, which has an assumed radix point which is fixed; and a part called the exponent which is the power of the radix by which the fraction is multiplied to produce the value represented. The radix is the number base for the representation; e.g., in scientific notation using a decimal number system, the radix is 10 and a number can be represented as

X * 10 ** Y

where X is the fractional part and Y the exponent. Rather than using 10 as a radix, computer systems generally use a radix of either 2, 8, or 16 to facilitate execution of FP operations on numbers stored as sequences of binary digits (bits). Where the radix is 2 (i.e., a binary FP representation), a number is represented as

X * 2 ** Y

and FP operations can be implemented efficiently. The three primary machine families examined herein use a binary representation; i.e., a radix of 2.

8. A general form for the internal representation of FP numbers can be viewed as follows:

±	exponent	±	fraction

A single bit is used for the algebraic sign of the fraction, an exponent sign bit is sometimes used (see paragraph 9 for an alternate exponent sign representation), and multiple bits are used for both the exponent and fraction magnitudes. In this conceptual view, moving from left to right in either the exponent or fraction fields is moving from most significant bit to least significant bit. Some representations place the assumed radix point to the left of the leftmost fraction bit, while others place it right of the rightmost bit. A common characteristic of representations is that FP values are "normalized"; i.e., numbers are stored with no insignificant leading zeros in the fraction so that the maximum number of significant digits may be stored. Normalization is easily accommodated since the value of the exponent determines the effective binary point, just as the value of the exponent determines the effective decimal point in scientific notation. The fraction can be readily adjusted to eliminate leading zeros, accompanied by a corresponding adjustment of the exponent. Thus, the size of the exponent field determines the range of numbers which can be represented, and the size of the fraction field determines the precision of the representation. FP precision can be characterized by the number of singificant digits which can be represented or by the machine "epsilon," or ε , which is defined as the smallest number such that $1. + \varepsilon > 1$.

A Sample Representation

9. Many different FP number representations are possible with differences in radix, location of radix point, treatment of exponent and fraction signs, size of exponent and fraction, and treatment of negative values. For purposes of exposition, a sample FP format will be presented based on a 32-bit word whose bits are numbered starting with bit zero at the left end as follows:

```
 0 1     7 8                        31
|s|xxxxxxx|yyyyyyyyyyyyyyyyyyyyyyyy|
```

where

s represents the sign of the fraction (0 is +, 1 is -)

y represents the exponent

x represents the fraction

In this format, the exponent field, rather than having an algebraic sign, is "biased" as a means of allowing for a negative exponent. A biased exponent is one for which a zero exponent value is represented by an exponent field containing a 1 for the leftmost bit and 0's for all others. For example, a zero exponent would be

1   7
1000000

The leftmost bit in the exponent field is considered the "bias bit," and the exponent would have a bias of 1 * 2 ** 6 = 64. Therefore, positive exponents would be represented by exponent fields containing values greater than 64, and negative exponents would correspond to exponent field values less than 64. Specifically, the exponent value for the sample format is equal to the content of the exponent field minus 64. An exponent of +5 would be represented as:

1   7
1000101

(69_{10})

and an exponent of -5 would be represented as:

1   7
0111011

(59_{10})

This sample format has a binary exponent; i.e., the radix is 2, or the exponent denotes the power of 2 by which the fraction is multiplied to obtain the value represented. The binary point is implicitly located to the left of the leftmost fraction bit, and the fraction field is a signed-magnitude representation (i.e., the fraction field for a negative number is the same as that for the same positive number); but the algebraic sign field is different. All values are "normalized"; i.e., numbers are stored with no leading zeros in the fraction so that the maximum number of significant digits may be represented. Note that a complement form rather than signed magnitude is used for the fraction in some machines such as the PRIME. The PRIME's use of complement form for fractions will be explained in Part III. Some examples of values represented in the sample FP format are as follows:

$1._{10} = .1_2 * 2^1 =$

0	1 7	8 31
0	1000001	100000000000000000000000

$-1._{10} = -.1_2 * 2^1 =$

0	1 7	8 31
1	1000001	100000000000000000000000

$.25_{10} = .1_2 * 2^{-1} =$

0	1 7	8 31
0	0111111	100000000000000000000000

$-.25_{10} = -.1_2 * 2^{-1} =$

0	1 7	8 31
1	0111111	100000000000000000000000

$525.5_{10} = .10000011011_2 * 2^{10} =$

0	1 7	8 31
0	1001010	100000110110000000000000

$-525.5_{10} = -.10000011011_2 * 2^{10} =$

0	1 7	8 31
1	1001010	100000110110000000000000

Basic FP Operation Algorithms

Addition and subtraction

10. These operations are accomplished by first aligning the radix point of the two operands to the same relative position, performing the required operation (addition or subtraction) on the fractions, then normalizing the result. Using the following notation:

x and y	=	operands
s	=	sum
e(x), e(y), and e(s)	=	exponents of x, y, and s, respectively
f(x), f(y), and f(s)	=	fractions of x, y, and s, respectively

an addition algorithm for the sample representation can be stated more specifically as follows:

a. If $e(x) \neq e(y)$, then select the operand with the smaller exponent and shift its fraction right, incrementing its exponent by 1 for each bit shifted, until $e(x) = e(y)$; i.e., the radix points are aligned.

b. Add the operand fractions to obtain the sum fraction; i.e.,

$$f(s) = f(x) + f(y)$$

One of three exception cases may then occur:

(1) Case 1: f(s) = 0. Then set e(s) to 0 (most negative value); this forces s to 0.

(2) Case 2: f(s) overflows. Then shift f(s) right 1 bit (shifting the overflow bit into the most significant position) and set $e(s) = e(s) + 1$.

(3) Case 3: After Case 2, e(s) overflows. Then set the magnitude of s to the largest value, maintaining the proper sign.

c. Normalize s; i.e., shift f(s) left until the most significant bit is 1, subtracting 1 from e(s) for each bit shifted; if e(s) underflows, set s = 0.

The subtraction algorithm is the same except that in step b the operand fractions are subtracted rather than added.

11. Note that when the exponents are different, loss of significance sometimes occurs in the operand with the smaller exponent since its fraction is shifted right and bits are lost as they are shifted out of the least significant bit position. The larger the difference in the order of magnitude of the operands, the greater the potential loss of significance in the smaller operand. This problem will be addressed further later in this part.

Multiplication and division

12. Multiplication and division operations employ the following mathematical relations:

```
    x = f(x) * 2 ** (e(x))
    y = f(y) * 2 ** (e(y))
x * y = (f(x) * 2 ** e(x)) * (f(y) * 2 ** e(y))
      = (f(x) * f(y)) * 2 ** (e(x) + e(y))
  x/y = (f(x) * 2 ** e(x))/(f(y) * 2 ** e(y))
      = (f(x)/f(y)) * 2 ** (e(x) - e(y))
```

Therefore, the multiplication operation, for example, can be accomplished by multiplying the fractions, adding the exponents, and normalizing the results. If p is the product and e(p) and f(p) are the exponent and fraction of the product, respectively, an algorithm for the sample representation can be stated more specifically as follows:

a. Multiply the operand fractions to obtain the double-length product fraction; i.e.,

f(p) = f(x) * f(y)

b. Add the operand exponents to obtain the product exponent; i.e.,

e(p) = e(x) + e(y) - bias

One of three exception cases may occur:

(1) Case 1: e(p) overflows. Then set f(p) to the largest magnitude fraction with the proper sign and set e(p) to the largest positive exponent.

(2) Case 2: e(p) underflows. Then set f(p) to 0 and set e(p) to 0 (most negative value); this forces p to 0.

(3) Case 3: f(x) or f(y) = 0. Then set f(p) to 0 and set e(p) to 0; this forces p to 0.

c. Normalize the double-length product (normalization may produce Case 2 above).

d. Round the product to the proper word length, renormalizing if required.

The division algorithm is similar and will not be stated here.

13. Note that the multiplication algorithm includes none of the fraction shifting that produces lost operand bits as in addition and subtraction. One might intuitively believe from this that addition and subtraction, rather than multiplication and division, are the major culprits in FP accuracy loss. This conclusion can, in fact, be shown to be true (Knuth 1969), and it is well established that substantial losses in accuracy can be expected from addition and subtraction in some cases, but not from multiplication and division. These losses are addressed in the next section.

Errors in Operations

Large differences in operand magnitudes

14. From the description of the addition algorithm presented above, it is apparent that addition or subtraction performed on two operands with large differences in magnitudes will result in loss of significant digits in the operand of lesser magnitude--perhaps even of the entire operand. Using the sample FP representation, the fraction is represented by 24 binary digits which is equivalent to less than 8 decimal digits. Hence, for the addition

$$40000. + .0025$$

the second operand is totally lost during the operation and the sum produced is simply 40000. However, our concern is generally with the relative error (i.e., the magnitude of the error relative to the true result) which in this case will be less than 1 in 10 ** 7, and in general will be on the order of 2 ** (-n) where n is the number of bits in the fraction. Therefore, in some cases where a single operation produces a final result, loss of significance due to magnitude differences in operands, such as shown in this example, may be unimportant. In other cases where an expression requires several operations in sequence, errors due to greatly varying operand magnitudes can produce unexpected results if the order in which operations are performed is not carefully selected. Specifically, the associative law and the distributive law can fail rather badly. (See Part IV for examples.)

Small differences in operands

15. FP subtraction of very nearly equal values (or addition of values with nearly equal magnitudes but opposite signs) can produce very large relative errors. Again, considering the sample representation with 7+ decimal digits precision, given

$$s = x - y$$

where

$x = 2.575242$

$y = 2.575231$

then

s = 0.000011 or 11 * 10 ** (-6)

which is only a 2 significant digit result.

16. However, the potential roundoff error in each of the original operands is approximately $.5 * 10^{-6}$, meaning that the subtraction has produced only 1 reliable significant digit from two 7-digit operands. Thus, the maximum relative error in this example is potentially very high; i.e., approximately 1/11. Improper expression construction can unnecessarily produce large relative errors when the associative or distributive law fails due to nearly equal operands. (Again, see Part IV for examples.)

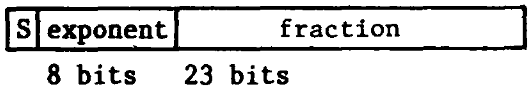
PART III: FLOATING-POINT REPRESENTATIONS FOR EACH MACHINE

The VAX 11 Family

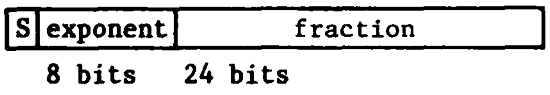
17. The VAX 11 FP representation takes advantage of the fact that for signed magnitude normalized numbers the high-order fraction bit is always 1 (see Part II), and thus in the stored value this bit is redundant and need not be kept. This feature allows, in effect, the gaining of an extra bit of significance from a given size stored fraction. To facilitate use of this concept, the hardware restores this "hidden" bit before performing arithmetic operations and likewise removes the bit before storing in memory the results of an operation. Thus, there are single- and double-precision representations for FP numbers in memory which are not identical with the corresponding representations in the arithmetic unit. Representations are shown below in a conceptual sense; i.e., fields constituting the values are depicted without showing the VAX numbering scheme for bits or byte addresses:

a. Single-precision:

(1) Value in memory:

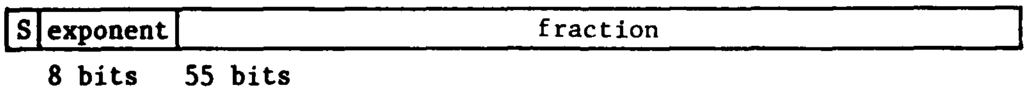


(2) Value as processed by arithmetic unit:

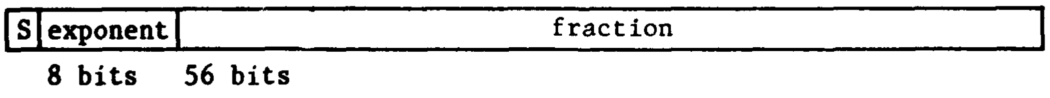


b. Double-precision:

(1) Value in memory:



(2) Value as processed by arithmetic unit:



18. Note that:

a. The sign bit applies to the fraction only.

b. The fraction has a signed magnitude representation.

c. As previously stated, the most significant fraction bit is not present in values in memory, but is restored by the hardware before arithmetic operations are performed.

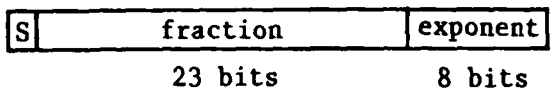
d. The exponent has a bias of 128_{10}; i.e., 200_8 or 10000000_2.

In the VAX's implementation of algorithms for FP operations, the arithmetic unit uses an "overflow" bit on the left and two "guard" bits on the right to ensure the return of a rounded result identical with the corresponding infinite-precision operation rounded to the specified fraction length; i.e., to ensure correct rounding. Thus, the rounded result of a FP operation has a roundoff error bound of half of the least significant bit of the fraction. Note that the 24-bit single-precision fraction is equivalent to approximately 7 decimal digits, that the 56-bit double-precision fraction is equivalent to approximately 16 decimal digits, and that the range for each is approximately $.29 * 10^{-38}$ through $1.7 * 10^{38}$.

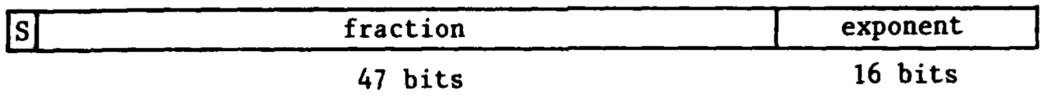
The PRIME 550 Series Family

19. The PRIME 550 Series uses memory representations of 32 bits for single-precision and 64 bits for double-precision as shown below:

a. Single-precision:



b. Double-precision:



20. Note that:

a. The sign bit applies to the fraction only.

b. The fraction uses a two's complement representation for negative numbers. In this case, "normalizing" the result of an operation means shifting the fraction left until the most significant bit differs from the sign bit, and the exponent is decreased by one for each bit shift.

c. There is no "hidden" or "understood" most significant fraction bit as in the VAX.

d. The exponent has a bias of 128_{10} (i.e., 200_8 or 10000000_2) for both single- and double-precision values in memory.

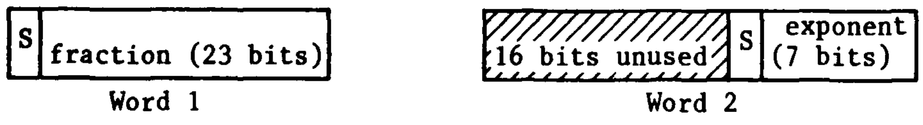
The FP register used by the arithmetic unit for single-precision operations has a 16-bit exponent and 31-bit fraction. Thus, the arithmetic unit can use longer fractions and larger exponents internally while executing the FP operation algorithms, and representation conversions must take place automatically as FP values move between the arithmetic unit and memory. While the added length of the arithmetic unit fraction certainly should allow correct rounding to 23 bits, roundoff does not take place automatically when a floating point value is stored from register to memory. Rather the extra 8 bits of fraction are truncated when a floating store is executed. In this case, the error bound is equal to the value of the least significant bit of the fraction. With double-precision operations, the FP register representation is the same as the memory representation, and the fraction is truncated at 47 bits.

21. One might question the utility of a 16-bit exponent in the FP register while only 8 bits are used for a stored value. The larger exponent allows the generation of larger magnitude values within the arithmetic unit, but an attempt to store such a value produces an exception condition, the processing of which could provide for retrieval of the excess magnitude value using special instructions. It is not known whether PRIME FORTRAN provides any facility for using single-precision values with 16-bit exponents, but there would seem to be little use for such a facility. The standard precision and range of values for the PRIME are approximately the same as those for the VAX with the exception that the 47-bit double-precision fraction is equivalent to approximately 14 decimal digits as opposed to 16 for the VAX.

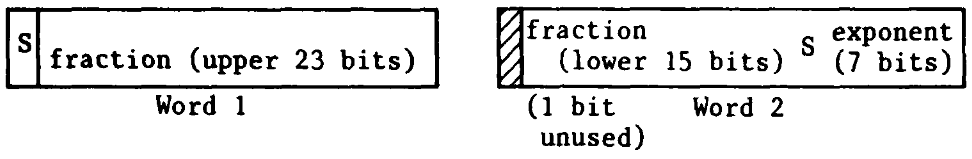
Harris Series 500 Systems

22. The Harris Series 500 Reference Manual (Harris Corporation 1978) specifies single-, double-, and quadruple-precision FP data formats as follows:

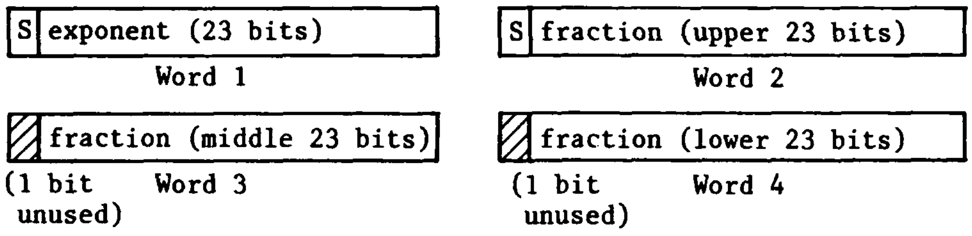
a. Single-precision (non-SAU machines):



b. Double-precision (and single-precision for SAU machines):



c. Quadruple-precision:



23. Note that:

a. Both the exponent and the fraction are signed.

b. Both the exponent and the fraction use a two's complement representation for negative numbers.

c. There is no "hidden" most significant fraction bit as in the VAX.

d. In each representation, a single "unused" bit in the high-order position of each fraction word after the first, although not used in the FP representation, is reserved for specific usage during execution of some instructions.

e. The results of all operations are truncated, not rounded.

24. Upon examination of the data formats, one might immediately question the value of the single-precision format since it uses the same amount of memory while providing significantly less precision. In fact, Harris documentation clearly indicates that for the Series 500 all FP operations performed by the Scientific Arithmetic Unit (SAU) (i.e., its FP hardware) are executed in the double-precision FP format. Thus, use of single-precision numbers would seem to be of value only in machines without the SAU option; i.e., those in which FP operations are constructed from sequences of integer operations ("software" FP). In this case, software implementations of single-precision operations could definitely execute significantly faster than those for double-precision operations. While it seems clear that use of single-precision values is prudent only for non-SAU equipped machines, the Harris FORTRAN Manual (Harris Corporation 1981) leaves one in doubt as to how FP data types are implemented for SAU-equipped machines. Specifically, the Harris FORTRAN

Manual shows the REAL data type to be the standard single-precision representation shown above unless the compile time option "P" is used which changes the REAL data type to the double-precision representation shown. But in fact, execution of test programs (detailed later) shows that, for SAU-equipped machines, the stated double-precision representation is actually used internally (both in memory and within the SAU) for all REAL values, regardless of the use of the "P" option. Therefore, the accuracy of computations is identical for single-precision and double-precision values. The only detected difference in the treatment of the two is in the input/output field lengths provided. Values implicitly or explicitly declared to be single-precision have input and output field lengths truncated to the specified single-precision length; e.g., the formatted output of a single-precision value is truncated to 7 significant decimal digits. Given that there is no significant advantage to using so-called "single-precision" values, it is probably wise to use the "P" compiler option as standard practice unless there is specific justification for not doing so. (Note that all existing Corps of Engineers' Harris 500 systems have the SAU option.)

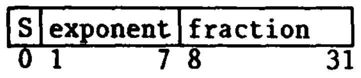
25. Harris also provides a quadruple-precision data type, as shown in paragraph 22, apparently implemented via software FP operations. Such software implementations of extended-precision data types are generally very slow, a fact which execution of the test programs confirmed, and are not candidates for widespread usage in Corps S&E applications. However, they can be valuable in conducting application accuracy studies and sometimes in production programs when extended-precision data items are carefully selected.

26. In summary, the Harris 500 Series, with optional SAU, provides a data type used for both single- and double-precision values which has a 38-bit fraction providing approximately 11 decimal digits, and a software implemented quadruple-precision data type providing approximately 20 decimal digits.

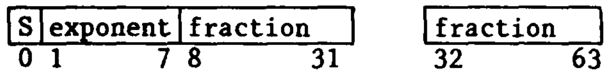
IBM 4331

27. The IBM 4331 FP representations, shown below, are used widely throughout the IBM product line. The FP accuracy for IBM systems is not the primary subject of this study and no explanation of the representation will be presented.

a. Single-precision:

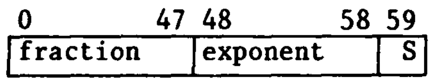


b. Double-precision:



CDC CYBER 6600

28. The CDC CYBER 6600 is a large-scale computer system with a memory word size of 60 bits. The FP representation for single-precision format is shown below. This system has been the subject of several accuracy studies (Ward 1976, O'Neil and Peterson 1976). The results of these studies have shown the CDC 6600 to be a highly accurate system for the Corps' S&E problems. (For this study, all test programs run on the CDC system used only single-precision FP values.)



Further FP Architecture Characteristics

29. Several machine accuracy characteristics which may be of interest (e.g., machine epsilon, largest and smallest FP numbers, etc.) can be determined from actual machine execution of FP operations. In Appendix B of Cody and Waite (1980) is a subroutine called MACHAR for determining 13 machine constants which allows one to check manufacturer's claims. Results of execution of this subroutine on each of the five test systems are presented in Appendix A.

PART IV: EXAMPLES OF FLOATING-POINT ERROR

30. Since FP values and the operations on such are approximations of real values and the corresponding exact operations, the fundamental laws related to operations on real numbers sometimes break down producing substantial and even dramatic errors. Examples of two such cases and of the effects of fraction length and treatment of rounding are given below.

Failure of the Associative Law of Addition

31. It is quite common practice in programming to apply the associative law of addition:

$$(x + y) + z = x + (y + z)$$

That is, in performing a sequence of additions, to produce a single sum, the order in which the additions are performed is usually not considered a matter for concern; i.e., associativity is assumed. This is not surprising since common mathematical notations for summation are inherently based on associativity. However, failure of the associative law can occur in FP operations; i.e.,

$$(x + y) + z \neq x + (y + z)$$

in some cases.

32. For example, consider a hypothetical FP representation with exactly 7 decimal digits; i.e., a true decimal representation. Then, the expression

$$\begin{aligned} &(5505026. + (-5505024.)) + 3.9375 \\ &= 2. + 3.9375 \\ &= \underline{5.9375} \end{aligned}$$

which is the exact result.

33. However, if we change the order of evaluation such that the expression becomes

$$\begin{aligned} &5505026. + (-5505024. + 3.9375) \\ &= 5505026. + (-5505021.) \\ &= 5.0 \end{aligned}$$

if truncation rather than rounding takes place, or

$$= 5505026. + (-5505020.)$$
$$= 6.0$$

if rounding takes place, neither result is exact. The associative law has failed in both cases, but where rounding was not used, the failure is much more pronounced.

34. Now consider a binary FP representation with a 23-bit fraction (such as those for systems examined herein) and the summation expression above. The values in the expression have fractions and exponents as follows:

	fraction	
(5505026.)	$\boxed{10101000000000000000010}$	exp = 23_{10}
(5505024.)	$\boxed{10101000000000000000000}$	exp = 23_{10}
(3.9375)	$\boxed{11111000000000000000000}$	exp = 2

As above, the first expression for the sum of the three values will produce the correct result and the second will not. The subexpression (-5505024. + 3.9375) again is of interest as the source of error. The addition algorithm (see Part II) shifts the fraction for 3.9375 some 21 bit positions right to align the binary point, resulting in the following:

	fraction	
(3.9375)	$\boxed{00000000000000000000011}$	exp = 23_{10}

35. In effect, 3.9375 has become 3.0 and the addition will produce -5505021., if truncation rather than rounding takes place. However, if additional fraction bits are employed within the arithmetic unit to achieve correct rounding, the result of the subexpression is -5505020., producing a complete expression result of 6.0. Thus, in this case, the result of truncation and rounding in the FP binary representation is much the same as we would expect if we examined a true decimal representation with the equivalent number of decimal digits. Such is not always the case, as illustrated by the next example of failure.

36. In the example above, single-precision truncation of results in the PRIME would produce an expression result of 5.0 rather than 6.0. The VAX, with the extra "hidden" bit and the facility for correct rounding, would produce 6.0. The Harris single-precision or use of double-precision in any of

the three machines would produce correct results; i.e., 5.9375. While this example was specifically chosen to illustrate a case where a 23-bit fraction is inadequate to support associativity, the principle holds for any size fraction.

Failure of the Distributive Law

37. Similarly the distributive law

$$x * (y + z) = (x * y) + (x * z)$$

is commonly assumed to be valid for FP operations, but in fact can fail rather badly in some cases. Consider again the hypothetical FP representation with exactly 7 decimal digits. Given that x = 20000., y = -6.0, and z = 6.000003,

$$\begin{aligned} x * (y + z) &= 20000. * (.000003) \\ &= \underline{.06} \end{aligned}$$

which is the correct result.

38. However, if we attempt to apply the distributive law,

$$\begin{aligned} (x * y) + (x * z) &= (20000. * -6.) + (20000. * 6.000003) \\ &= -120000. + 120000.1 \\ &= .1 \end{aligned}$$

if correct rounding takes place, or

$$\begin{aligned} &= -120000. + 120000.0 \\ &= 0. \end{aligned}$$

if truncation takes place. Thus, the failure using a true decimal representation in this case could be disastrous; e.g., if truncation takes place and the result is used as a divisor.

39. However, true decimal representations are rarely used for FP arithmetic (it is shown here only for illustrative purposes). Therefore, let us consider this example using a binary representation with a 23-bit fraction, as in the previous example. It can be shown (the multiplication is lengthy

and will be omitted) that for such a representation, the distributed expression above produces results of

$$.0625$$

if correct rounding is achieved, or

$$.0469$$

if truncation is used. Again, neither result is the true value which is produced by the first expression. Thus, the distributive law has failed; i.e.,

$$x * (y + z) \neq (x * y) + (x * z)$$

While the magnitude of the error is small, it must be remembered that it is the relative error that is significant, and the relative error here is large. It is also significant to note the large increase in the relative error if rounding is not used; i.e., for rounding

$$\text{relative error} = \frac{(.0625 - .06)}{.06} = \frac{.0025}{.06}$$

$$= 4^{+}\%$$

for truncation,

$$\text{relative error} = \frac{(.06 - .0469)}{.06} = \frac{.0131}{.06}$$

$$= 21^{+}\%$$

Such errors which are small in an absolute sense but large in a relative sense can easily produce large magnitude errors when the results are used in further computations; e.g., if the result above is used as a divisor with a large dividend, the absolute error produced could be very large.

40. Again this example was chosen to illustrate a severe case, but such cases could easily be produced in real programs.

PART V: TEST RESULTS

41. Two sets of test problems were run on the computers selected for this study. The problems were chosen carefully to ensure that they would provide information on accuracy of the computers. The first set is a group of mathematical problems. The problems strain each system to a considerable extent in its ability to handle computations. A simple example of adding a series of numbers forward and backward is followed by an evaluation of a polynomial function and the inversion of a simple but intriguing matrix. Complete listings of the mathematical programs used are included in Appendix B.

42. The second set of examples was chosen from Corps of Engineers files. This set represents real world conditions, and the problems and data chosen were what one could reasonably be expected to encounter in design and/or analysis applications. The first problem is an analysis of a flexible sheet pile bulkhead using the finite difference approach. The second problem uses the finite element technique for analysis of a soil-structure interaction problem of a sheet pile embedded in soft clay. The last problem involves computing the buckling load of a pile using again the finite difference procedure. These problems were selected to show that engineers must be careful in using computer programs in machines that do not carry a sufficient number of significant digits in computations.

Mathematical Problems

Addition of a series of integers--program MAXJ

43. The first test program is very short and simple but demonstrates how accuracy can be lost in simple arithmetic operations.

44. One of the basic arithmetic laws, the commutative law of addition, says that

$$a + b = b + a$$

i.e., in adding two numbers, it does not matter which is placed first. This law holds rigorously for arithmetic in which all numbers are exact, even when extended to apply to a sequence of an arbitrary number of additions. However, for a sequence of an arbitrary number of FP additions, the law does not necessarily hold.

45. Program MAXJ, which is given a maximum value in J, will calculate a forward and backward sum of the positive integers from 1 through J. The program first computes the forward sum "SL" in which

$$SL = 1. + 2. + 3. + \ldots + \text{Float}(J-1) + \text{Float}(J)$$

and the backward sum "SU" in which

$$SU = \text{Float}(J) + \text{Float}(J-1) + \text{Float}(J-2) + \ldots +2. +1.$$

MAXJ then calculates the difference in the two sums as "DIFF."

46. All computers have a maximum FP value, determined primarily by their maximum exponent value. Thus, in computing the sum of a sequence of integers from 1 to J, any system's FP value will overflow at some value of J. Computers that have larger exponent fields can be expected to overflow at a larger value of J than those with a small exponent field. If FP number representations were exact for all sums less than a machine's maximum FP value, then it would be possible to calculate DIFF in the forward and backward sums, as shown above, and DIFF would always be zero so long as no FP overflow occurred. However, FP representations are approximate not only for fractional values but also for integers whose magnitudes exceed the number of digits which can be represented exactly. Such approximations occur frequently since it is quite common for FP representations to accommodate magnitudes of 10 ** 38 but only store the equivalent of approximately 7 significant decimal digits; e.g., in the VAX and PRIME representations. Where sums exceed the maximum value which can be represented exactly, forward summing will produce roundoff errors different from those produced by backward summing, and thus nonzero values of DIFF will occur long before the occurrence of a FP value overflow. The results of program MAXJ, summarized in Table 1, illustrate this condition. Program runs were made on each system using each available precision with J = 10^3, 10^4, 10^5, 10^6, and 10^7 in that order. When a given value of J produced a nonzero value of DIFF, no further runs were made on that system at that precision. Only single-precision was tested on the CDC. The Harris and the CDC were the only systems in which DIFF was 0 with J as large as 10^5 in single-precision. The other systems failed beyond J = 10^3. In double-precision, the Harris failed beyond 10^5, as in single-precision, but

Table 1

Summary of MAXJ Runs

System	Precision	Computed Results for Cited J				
		10^3	10^4	10^5	10^6	10^7
CDC	Single	0	0	0	99999.	NR*
Harris	Single	0	0	0	20630.	NR
	Double	0	0	0	20630.	NR
	Quadruple	0	0	0	0	0
IBM	Single	0	-29552.	NR	NR	NR
	Double	0	0	0	0	0
PRIME	Single	0	-13408.	NR	NR	NR
	Double	0	0	0	0	0
VAX	Single	0	1972.	NR	NR	NR
	Double	0	0	0	0	0

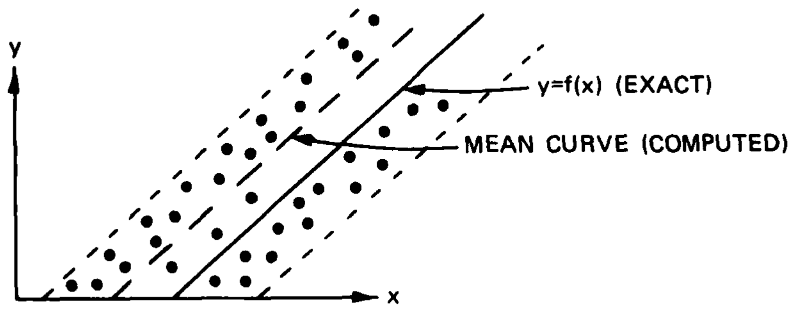
* Not run.

all the other minisystems were able to compute up to 10^7.

Evaluation of a polynomial function--program POLY

47. This example was run to obtain some "overall" comparisons between the systems. First, however, the concept of "noise level" will be explained.

48. Let us suppose that a continuous function defined by y = f(x) needs to be plotted. In a computer, to plot y versus x, y is evaluated for various values of x. Since the function is continuous, one would expect to obtain a nice smooth continuous curve for y. However, if the function is complicated, due to rounding errors, a scatter of values could result, as illustrated below in an exaggerated manner.



The computed function would lie in the band indicated by the dotted lines. One could also produce a "mean curve" by computing not simply a single value of y for each selected value of x, but the mean of values of y computed at x and at several points on either side of x. If the arithmetic were exact (i.e., infinitely precise) and the points on either side of x were chosen to be very close to x, then the computed mean curve would coincide very nearly to the exact function curve. However, given the limited precision of FP representations, this mean curve will not necessarily coincide exactly with the function y = f(x); i.e., there will be a "bias." The computed points will be scattered around the mean curve. A convenient measure of this scatter has been chosen to be twice the standard deviation from the mean curve and has been designated as the "noise level" (Noble 1982).

49. We will now evaluate a particular example using this concept. The polynomial equation whose roots are the first 20 integers is given below:

$$\begin{aligned} P_{20}(x) &= (x - 1)(x - 2) \ldots (x - 20) \\ &= x^{20} + a_1 x^{19} + a_2 x^{18} + \ldots + a_{20} \end{aligned}$$

This function has roots that are very sensitive to small variations in the coefficients of the higher powers of x. This example may be too ill conditioned for the smaller systems, so we will use a program called POLY which uses a polynomial whose roots are the first 11 integers:

$$\begin{aligned} P_{11}(x) &= (x - 1)(x - 2) \ldots (x - 11) \\ &= x^{11} + a_1 x^{10} + a_2 x^{9} + \ldots + a_{11} \end{aligned}$$

Figure 1 shows a plot of this function when $P_{11}(x)$ is evaluated for various values of x (taken from Noble 1982).

50. The evaluation of $P_{11}(x)$ at the smaller x values shows small deviations (noise level), but as the x values increase we can see an increase in the noise level.

51. Program POLY first computes the coefficients of the polynomial, a_i, i = 1 ... 11. (Computation of the a_i values on each of the systems produced identical results in all precisions.) It then evaluates mean and standard deviation values of $P_{11}(x)$ for selected values of x; i.e., it in effect produces the mean curve and standard deviations from the mean at selected points on the curve. Mean and standard deviation values of $P_{11}(x)$ are computed from a set of values obtained by computing, by nested multiplication,

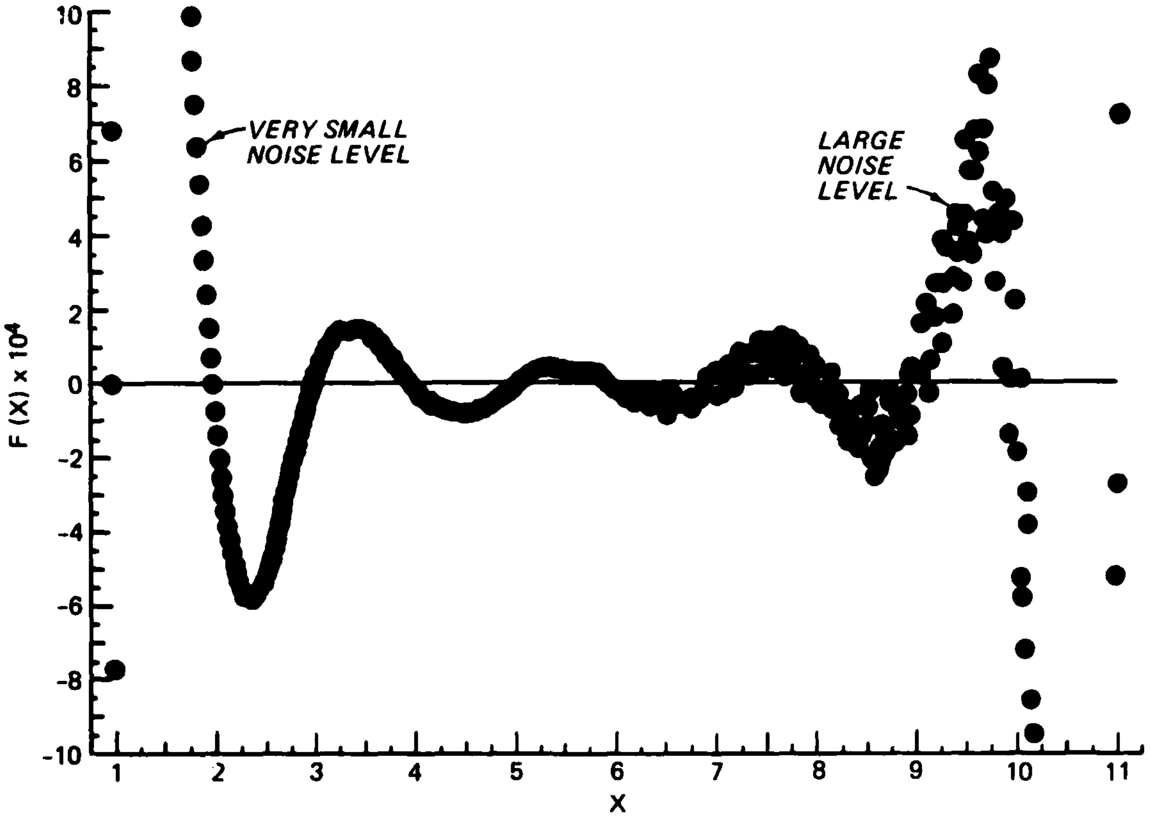


Figure 1. Plot of a polynomial function of degree 11

$P_{11}(z)$ for $z = x + i\varepsilon$ for $i = -m, -m+1, \ldots m-1, m$ (where x, m, and ε are supplied by the user); i.e., POLY computes $2m + 1$ values of $P_{11}(z)$ for each selected value of x and then takes the mean and standard deviation of these $2m + 1$ values.

52. The theoretical mean of values around a root of a function should be close to zero if m and ε values are small (since $P_{11}(x) = 0$ at the roots). Computed values of the mean and standard deviation at roots can be used as metrics of the relative accuracy performance of the various systems.

53. The same set of data was supplied to each system. Parameters used were $m = 2$ and $\varepsilon = 1.E-7$. The mean and standard deviation of $P_{11}(x)$ were computed at each root of the polynomial; i.e., at $x = 1, 2, \ldots, 10, 11$. Tables 2-5 present the results from the runs on all of the systems.

54. It is obvious from the results that different systems compute different mean and standard deviation values. This effect is primarily due to differences in FP precision. When we examine the standard deviation, we can

Table 2

POLY-Computed Means of $P_{11}(x)$ in Single-Precision

x	Mean Computed by Cited System				
	CDC	Harris	IBM	PRIME	VAX
1.0	*	*	16	1.2	-5
2.0	*	*	512	120.	36
3.0	*	*	111	360.	-158
4.0	*	-.1	1555.	1476.	302
5.0	*	-.2	2214.	3358.	-1000
6.0	*	-.4	-6749.	2035.	-2188
7.0	*	-1.1	12602.	22776.	-4704
8.0	*	-1.7	11846.	45353.	-6656
9.0	*	-6.4	85098.	51174.	-8554
10.0	*	-16.1	146330.	272236.	-36800
11.0	*	-21.4	214960.	65844.	-79762

* Value less than ±0.05.

Table 3

POLY-Computed Means of $P_{11}(x)$ in Double- and Quadruple-Precision

x	Mean Computed by Cited System in Cited Precision				
	Harris in		IBM in	PRIME in	VAX in
	Double	Quadruple	Double	Double	Double
1.0	*	*	*	*	*
2.0	*	*	*	*	*
3.0	*	*	*	*	*
4.0	-.1	*	*	*	*
5.0	-.2	*	*	*	*
6.0	-.4	*	*	*	*
7.0	-1.1	*	*	*	*
8.0	-1.7	*	*	*	*
9.0	-6.4	*	*	*	*
10.0	-16.1	*	*	*	*
11.0	-21.4	*	*	*	*

* Value less than ±0.05.

Table 4

POLY-Computed Standard Deviations of $P_{11}(x)$ in Single-Precision

	Standard Deviation Computed by Cited System				
x	CDC	Harris	IBM	PRIME	VAX
1.0	1.18	1.18	4.90	2.60	3.02
2.0	.08	.08	85.90	121.26	12.33
3.0	.18	.19	354.20	85.43	47.55
4.0	.01	.01	1105.50	1173.10	184.25
5.0	.04	.22	7430.90	3198.46	1155.51
6.0	.03	.07	20396.60	9619.22	462.30
7.0	.04	.19	56699.00	15829.48	12721.25
8.0	.07	3.90	179411.00	133981.78	15684.96
9.0	.18	4.00	64710.00	146073.44	38611.73
10.0	.81	4.83	167855.00	416287.44	89089.88
11.0	8.08	21.27	242729.00	686520.50	124516.00

Table 5

POLY-Computed Standard Deviations of $P_{11}(x)$

in Double- and Quadruple-Precision

	Standard Deviation Computed by Cited System in Cited Precision				
	Harris in		IBM in	PRIME in	VAX in
x	Double	Quadruple	Double	Double	Double
1.0	1.18	1.18	1.18	1.18	1.18
2.0	.08	.08	.08	.08	.08
3.0	.19	.18	.18	.18	.18
4.0	.01	.01	.01	.01	.01
5.0	.22	.04	.04	.01	.04
6.0	.07	.03	.03	.01	.03
7.0	.19	.04	.04	.08	.04
8.0	3.90	.07	.07	.01	.07
9.0	4.0	.07	.07	.01	.07
10.0	8.83	.81	.81	.66	.81
11.0	21.27	8.11	8.1	8.4	8.4

note that the noise level of the Harris in single-precision increases for the larger values of x, but much less than the increase in noise levels of the IBM, PRIME, and VAX single-precision runs. The noise levels of the IBM, PRIME, and VAX in double-precision are smaller than that of the Harris in double-precision runs, and very close to those of the CDC and Harris quadruple-precision runs. As expected, the results of POLY show the Harris to be more accurate in single-precision than the IBM, VAX, or PRIME but less accurate than these systems in double-precision.

55. It should be noted that this is a severe test of accuracy; i.e., the "noise" produced for the higher values of x is extremely sensitive to differences in FP precision. Any machine can be made to perform badly in this test by choosing polynomials of higher and higher order. However, the order of the polynomial and the values of m and ε were chosen with the objective of producing a reasonable test of the relative accuracy of the machines in question.

Inversion of a matrix--program MATRIX

56. The last "textbook" test problem involves a numerical analysis program that will invert the Hilbert matrix using Gaussian elimination. This problem has been used in many studies on accuracy because:

a. Gaussian elimination is a straightforward, commonly used procedure for inverting matrices.

b. The Hilbert matrix is numerically unstable due to the "closeness" of the numbers.

c. This matrix has a known inverse against which we can compare the results (Ward 1976).

57. The Hilbert matrix is an n × n matrix of the following general form:

$$H_{ij} = \frac{1}{i + j - 1}$$

For example, if n = 2, then

$$H_{2\times2} = \begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{3} \end{bmatrix}$$

Or if n = 3, then

$$H_{3\times 3} = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

58. The inverse of a matrix A is defined as a matrix A^{-1} such that

$$A * A^{-1} = I$$

where I is the identity matrix of 1's on the main diagonal and 0's elsewhere. For example, if n = 2, then

$$H^{-1}_{2\times 2} = \begin{bmatrix} 4 & -6 \\ -6 & 12 \end{bmatrix}$$

59. Ward (1976) gives a general algorithm of the Gaussian elimination method. The algorithm is as follows:

a. $i \leftarrow 1$.

b. $p \leftarrow H_{ii}$ and $H_{ii} \leftarrow 1$.

c. Divide row i by p.

d. $Q_{ij} \leftarrow H_{ij} \leftarrow 1$ for all j such that $j \neq i$ and $1 \leq j \leq n$.

e. Multiply row i by Q_{ji} and subtract this product from row j for all j such that $j \neq i$ and $1 \leq i \leq n$.

f. $i \leftarrow i + 1$.

g. If $i \leq n$, then go to step b. Otherwise, stop here. H now contains the inverse of the $n \times n$ Hilbert matrix.

60. Program MATRIX was first executed on each system to compute the inverse of a 2 × 2 Hilbert matrix to verify the validity of the program. (Note: the input values used in the program are not the exact values of the imput matrices because the computers rounded or truncated the values to their greatest number of significant digits.) The computed inverse matrix was subtracted from the known exact inverse matrix (Figure 2) to obtain an approximate absolute error matrix. The error matrices for all the systems were very small. Next, a 6 × 6 Hilbert matrix (see Figure 3) was run on each of the systems. The error matrices for each system are shown in Figures 4-8 (the

$$\begin{bmatrix} 36 & -630 & 3360 & -7560 & 7560 & -2772 \\ -630 & 14700 & -88200 & 211680 & -220500 & 83160 \\ 3360 & -88200 & 564480 & -1411200 & 1512000 & -582120 \\ -7560 & 211680 & -1411200 & 3628800 & -3969000 & 1552320 \\ 7560 & -220500 & 1512000 & -3969000 & 4410000 & -1746310 \\ -2772 & 83160 & -582120 & 1552320 & -1746310 & 698544 \end{bmatrix}$$

Figure 2. Exact analytical solution of a 6 × 6 Hilbert matrix

$$\begin{bmatrix} \frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \\ \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} \end{bmatrix}$$

Figure 3. 6 × 6 Hilbert matrix

$$\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}$$

Figure 4. CDC approximate absolute error matrix
(* indicates value less than 1.0 E-6)

$$\begin{bmatrix} * & * & * & .1 & .1 & * \\ * & .1 & .7 & 1.7 & 1.9 & .7 \\ * & .7 & 4.4 & 11.6 & 12.8 & 5.1 \\ .1 & 1.7 & 11.5 & 30.1 & 33.3 & 13.2 \\ .1 & 1.9 & 12.8 & 33.3 & 36.8 & 64.5 \\ * & .7 & 5.0 & 13.1 & 64.5 & 5.7 \end{bmatrix}$$

a. Single- and double-precision

$$\begin{bmatrix} * & * & * & * & * & * \\ * & .1 & .3 & .9 & 1. & .4 \\ * & .3 & 2.3 & 6.1 & 6.7 & 2.6 \\ * & .9 & 6.1 & 15.8 & 17.4 & 6.9 \\ * & 1. & 6.7 & 17.4 & 19.3 & 57.6 \\ * & .4 & .3 & 6.9 & 57.6 & 3. \end{bmatrix}$$

b. Quadruple-precision

Figure 5. Harris approximate absolute error matrices
(* indicates value less than 0.05)

$$\begin{bmatrix} 5.5 & 154.5 & 1034.5 & 2671.8 & 2933.6 & 1151.2 \\ 153.7 & 4311.3 & 28846.7 & 74455.0 & 81715.1 & 32057.2 \\ 1025.3 & 28744.3 & 192224.4 & 495963.9 & 54418.3 & 213443.6 \\ 2641.1 & 74000.4 & 494695.5 & 127607.4 & 139989.4 & 549005.2 \\ 2894.2 & 81058.2 & 341741.2 & 654081.0 & 153257.5 & 600934.0 \\ 1134.1 & 31751.1 & 311796.7 & 158732.5 & 269860.6 & 220955.7 \end{bmatrix}$$

a. Single-precision

$$\begin{bmatrix} 1.6 & 45.4 & 302.4 & 778.3 & 852.9 & 334.3 \\ 45.4 & 1260.5 & 8345.2 & 21620.7 & 23702.8 & 92936.6 \\ 302.4 & 8395.2 & 55941.5 & 144133.8 & 158072.6 & 61998.1 \\ 778.3 & 21620.7 & 144133.8 & 371498.5 & 407547.6 & 159884.9 \\ 852.9 & 23702.8 & 158072.6 & 753045.2 & 447203.1 & 175426.8 \\ 334.3 & 9293.7 & 461961.9 & 197455.5 & 312320.3 & 237598.8 \end{bmatrix}$$

b. Double-precision

Figure 6. IBM approximate absolute error matrices

$$\begin{bmatrix} .4 & 11.6 & 79.6 & 209.4 & 233.1 & 92.5 \\ 11.9 & 348.9 & 2397.2 & 6300.9 & 7007.2 & 2776.7 \\ 83.8 & 2447.9 & 16799.4 & 44116.0 & 49026.2 & 19416.0 \\ 223.9 & 654.9 & 44770.5 & 117491.5 & 136502.5 & 51663.2 \\ 251.9 & 734.1 & 50299.0 & 131935.0 & 146470.0 & 58025.0 \\ 100.7 & 2933.2 & 20087.6 & 52670.0 & 58513.7 & 23132.4 \end{bmatrix}$$

a. Single-precision

$$\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}$$

b. Double-precision

Figure 7. PRIME approximate absolute error matrices
(* indicates value less than 0.05)

$$\begin{bmatrix} .5 & 14.2 & 94.9 & 244.9 & 268.8 & 105.4 \\ -14.1 & 397.8 & 2661.8 & 6868.1 & 7534.3 & 2954.3 \\ 94.8 & 2661.8 & 17800.9 & 45912.5 & 50351.7 & 19738.9 \\ 244.9 & 6868.0 & 45912.5 & 118385.4 & 129804.4 & 50877.7 \\ 268.8 & 7534.3 & 50351.7 & 129804.4 & 142301.8 & 55719.1 \\ 105.4 & 2954.3 & 19738.9 & 50877.7 & 55719.1 & 21854.2 \end{bmatrix}$$

a. Single-precision

$$\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}$$

b. Double-precision

Figure 8. VAX approximate absolute error matrices
(* indicates value less than 0.05)

results have been rounded to 1 decimal place).

61. The CDC, the baseline system, has a very small absolute error matrix. The Harris is the only system with a small error matrix in single-precision. The double-precision approximate error matrices for the PRIME and VAX were smaller than that for the Harris in quadruple-precision. The IBM had large error matrices in both single- and double-precision.

Real World Problems

62. For the S&E problems in this study, three computer programs were selected that have been in use for several years and have been thoroughly validated. These programs solve certain types of soil-structure interaction problems that are complex and for which no simple closed-form solution exist. Often the algorithm employed in the solution of a soil-structure interaction problem will yield incorrect results if the number of significant figures carried in the calculation is insufficient.

63. Three problems were chosen to show that different results can be obtained when identical problems are run on each of the systems tested using the same program. The problems have already been used in a previous study (O'Neil and Peterson 1976) with the same programs on a CDC system. The reported solutions from the CDC were duplicated in this study. The conclusions from the previous study indicated that the CDC's solutions were acceptable; therefore, these solutions are used for comparison with the results from the systems tested.

Case 1--flexible sheet pile bulkhead in sand

64. The deflections and moments produced in a flexible sheet pile bulkhead embedded in sand are to be computed for the physical system depicted in Figure 9. The problem was solved by using a computer code (BMCOL 28) which establishes the finite difference equations for a beam on an elastic subgrade at predesignated equally spaced nodes along the bulkhead (Matlock and Ingram 1963). The system of linear difference equations so generated forms a matrix equation in which the stiffness matrix is tightly banded. Recursive techniques are used to solve for the deflection at each node, and moments, shears, and soil reactions are subsequently computed.

65. It is common practice to analyze problems like that shown in Figure 9 by utilizing 50 to 100 nodes, but such solutions may be relatively

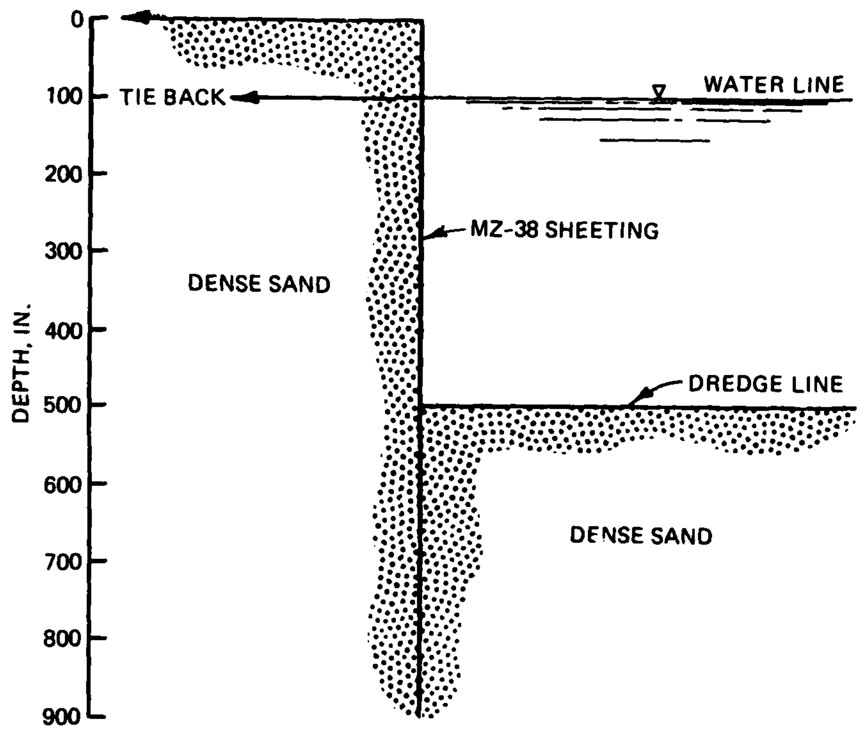
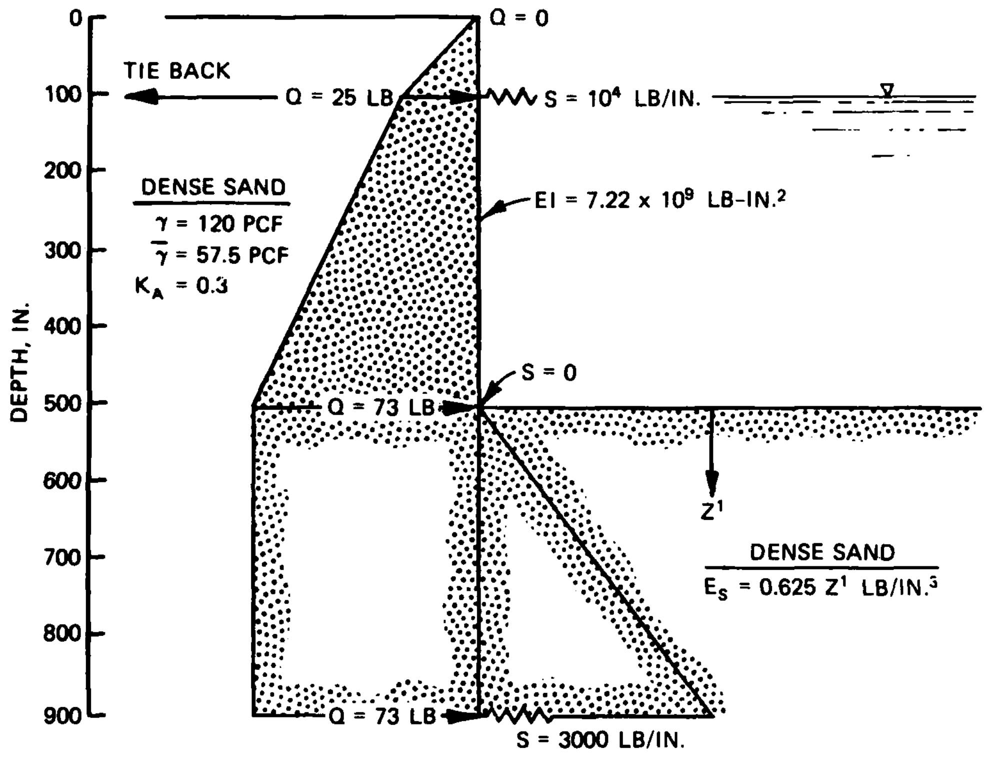


Figure 9. Physical system for Case 1

crude, especially if rapid moment gradients occur. Improved solutions can be obtained by increasing the number of nodes; i.e., by decreasing the increment length of finite spacing between nodes. However, as the increment length decreases below some value, the difference in deflection between two adjacent nodes decreases, which results in poorly defined derivatives and hence in an invalid solution. The increment length at which unrealistic answers are output is a function of the physical parameters input and of the number of significant digits used in the machine calculations.

66. Soil and bulkhead parameters input for this problem are shown in Figure 10. Specific numerical values given are for a solution using 900 increments. Identical runs were made with 9, 100, 225, 300, 400, 450, and 900 increments. The results are summarized in Figures 11-16.

67. The solutions from the IBM, PRIME, and VAX in single-precision are inconsistent and deteriorate as the number of increments increases. The Harris single-precision runs were identical with the Harris double-precision runs. All double-precision runs on all of the systems were virtually identical with the solution given by the CDC, with the exception that the Harris was off



NOTE: Q = IMPOSED LOAD/INCREMENT
S = SPRING CONSTANT FOR ONE INCREMENT
ALL VALUES FOR INCREMENTAL LENGTH OF 1 IN.
1-FT-WIDE SECTION

Figure 10. Soil and bulkhead parameters for Case 1

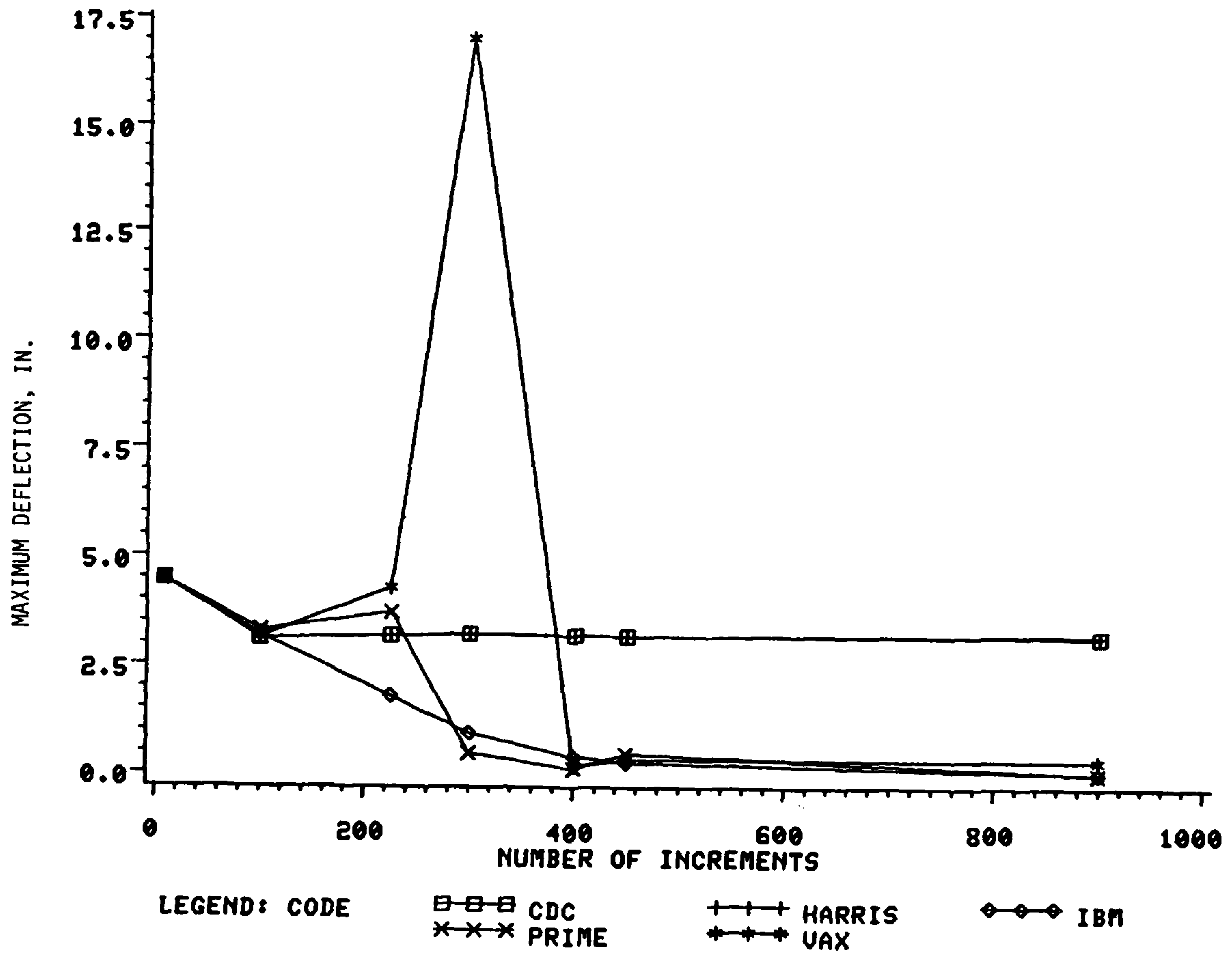


Figure 11. Maximum deflection versus number of increments, Case 1, single-precision

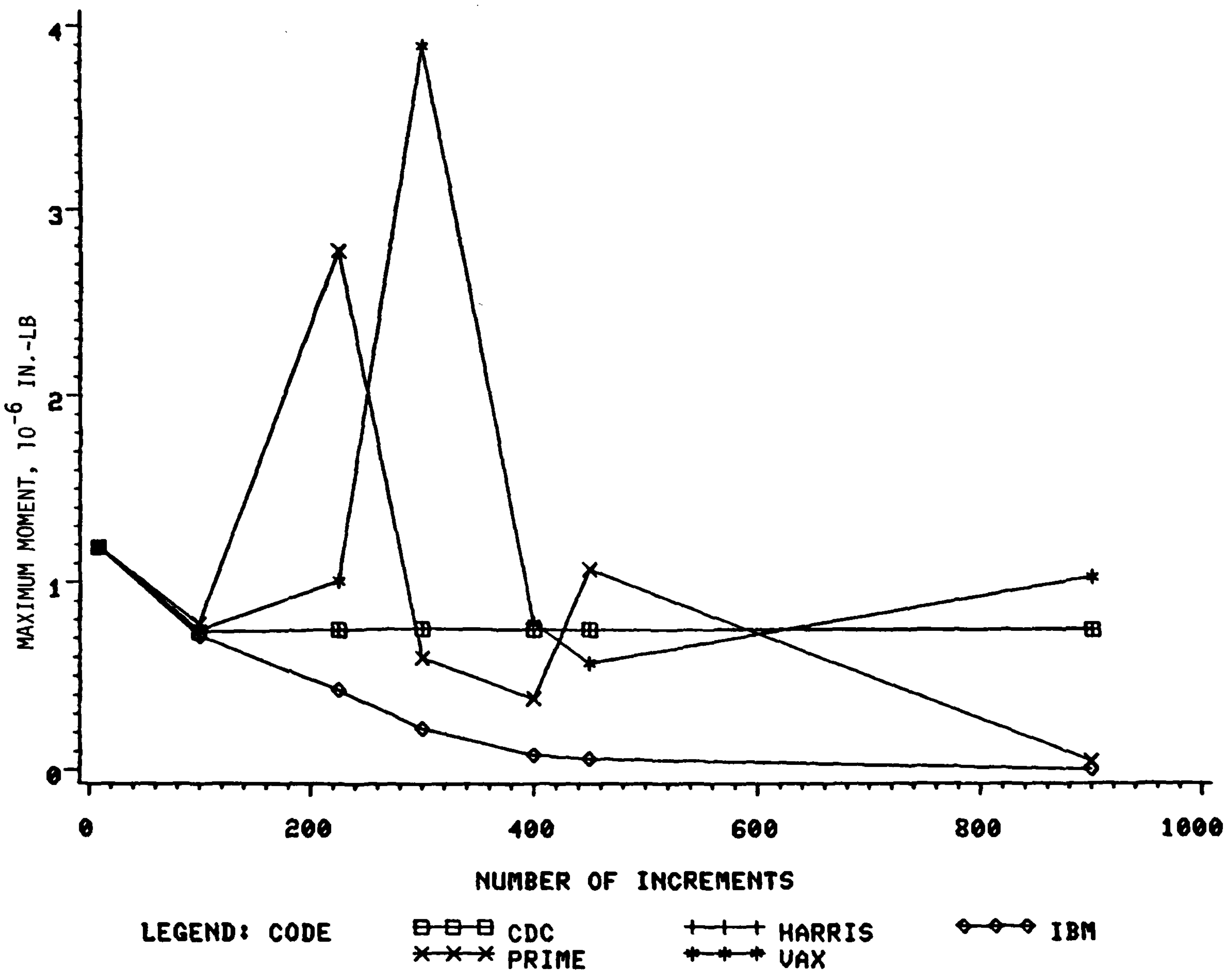


Figure 12. Maximum moment versus number of increments, Case 1, single-precision

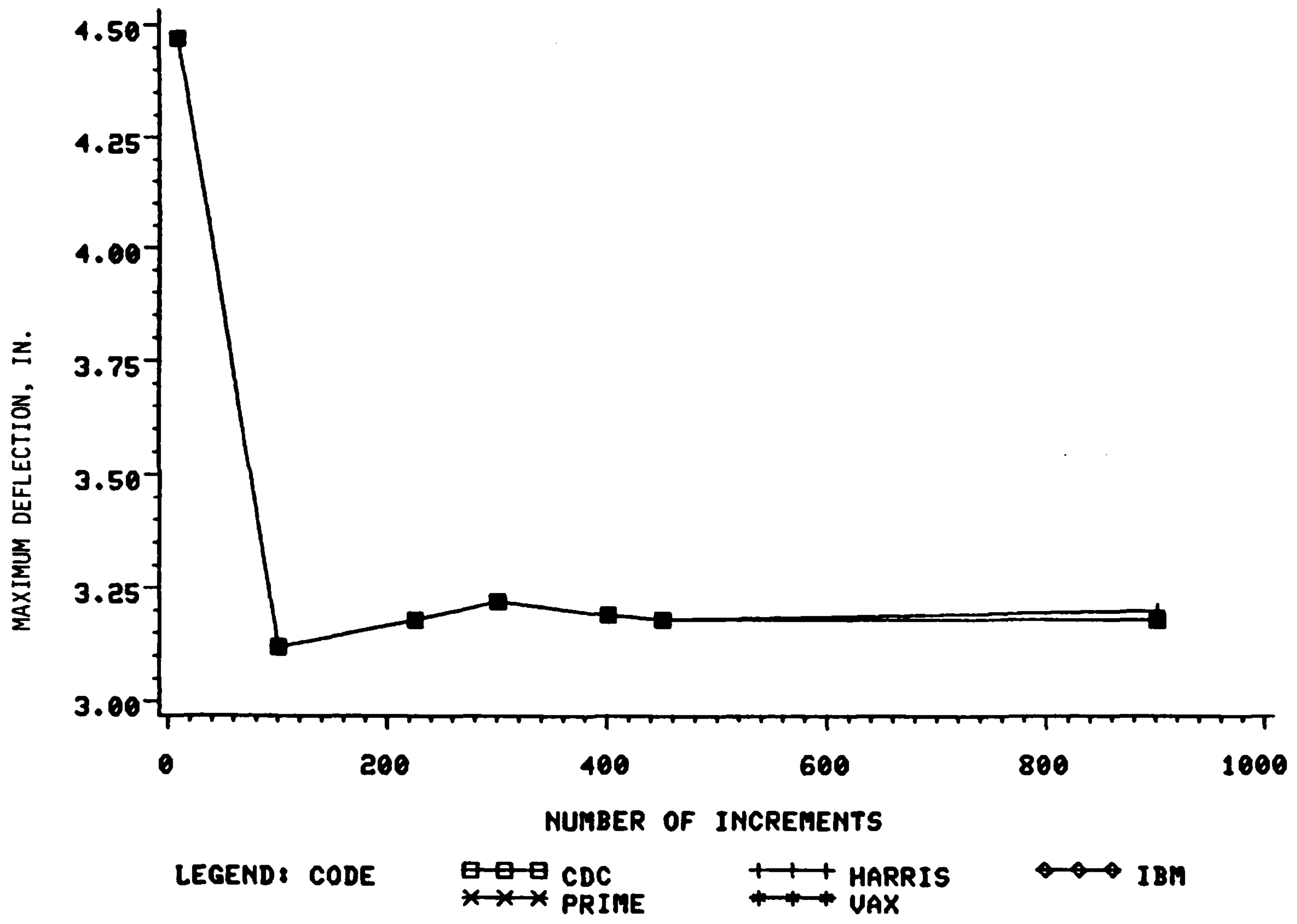


Figure 13. Maximum deflection versus number of increments, Case 1, double-precision

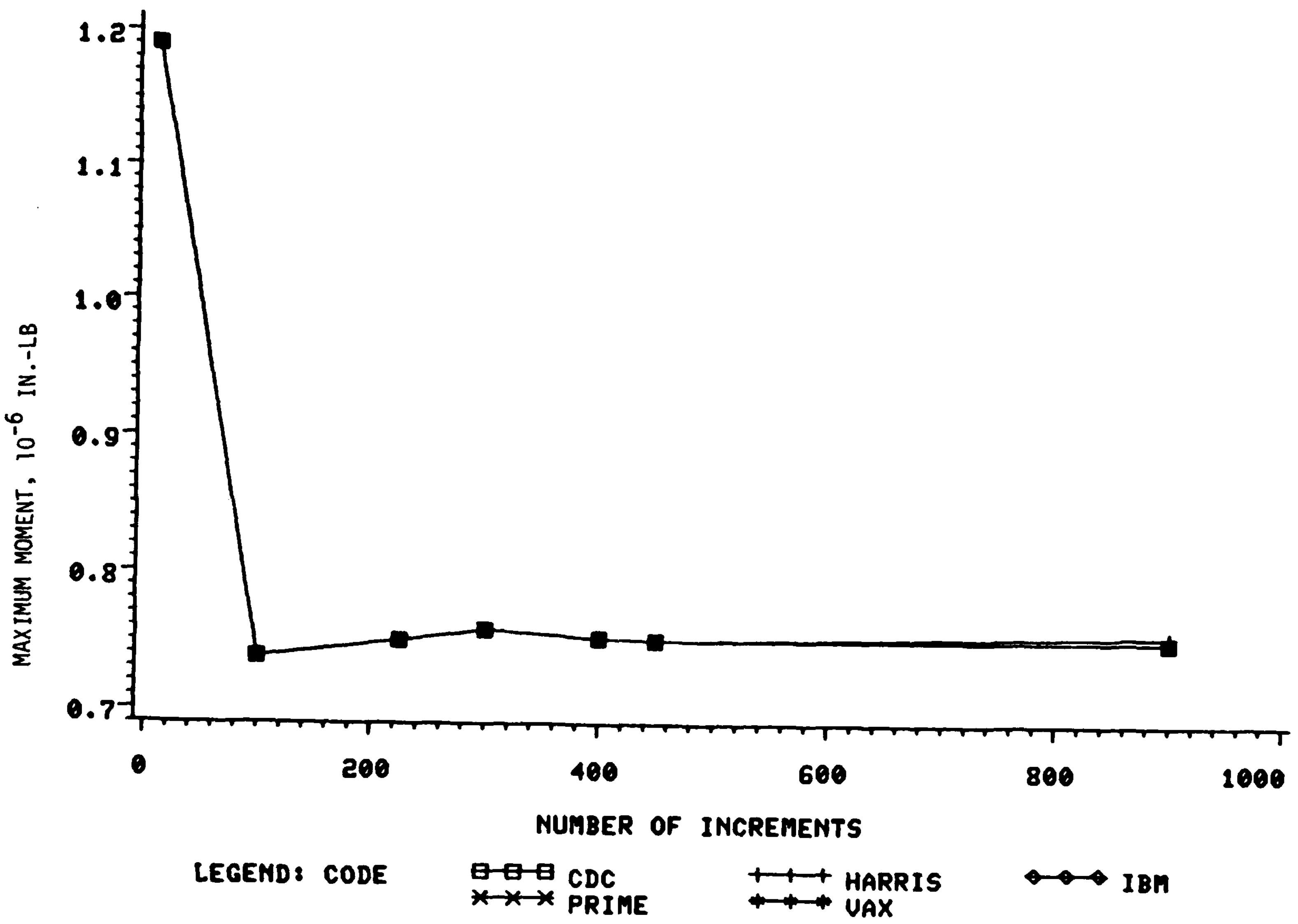


Figure 14. Maximum moment versus number of increments, Case 1, double-precision

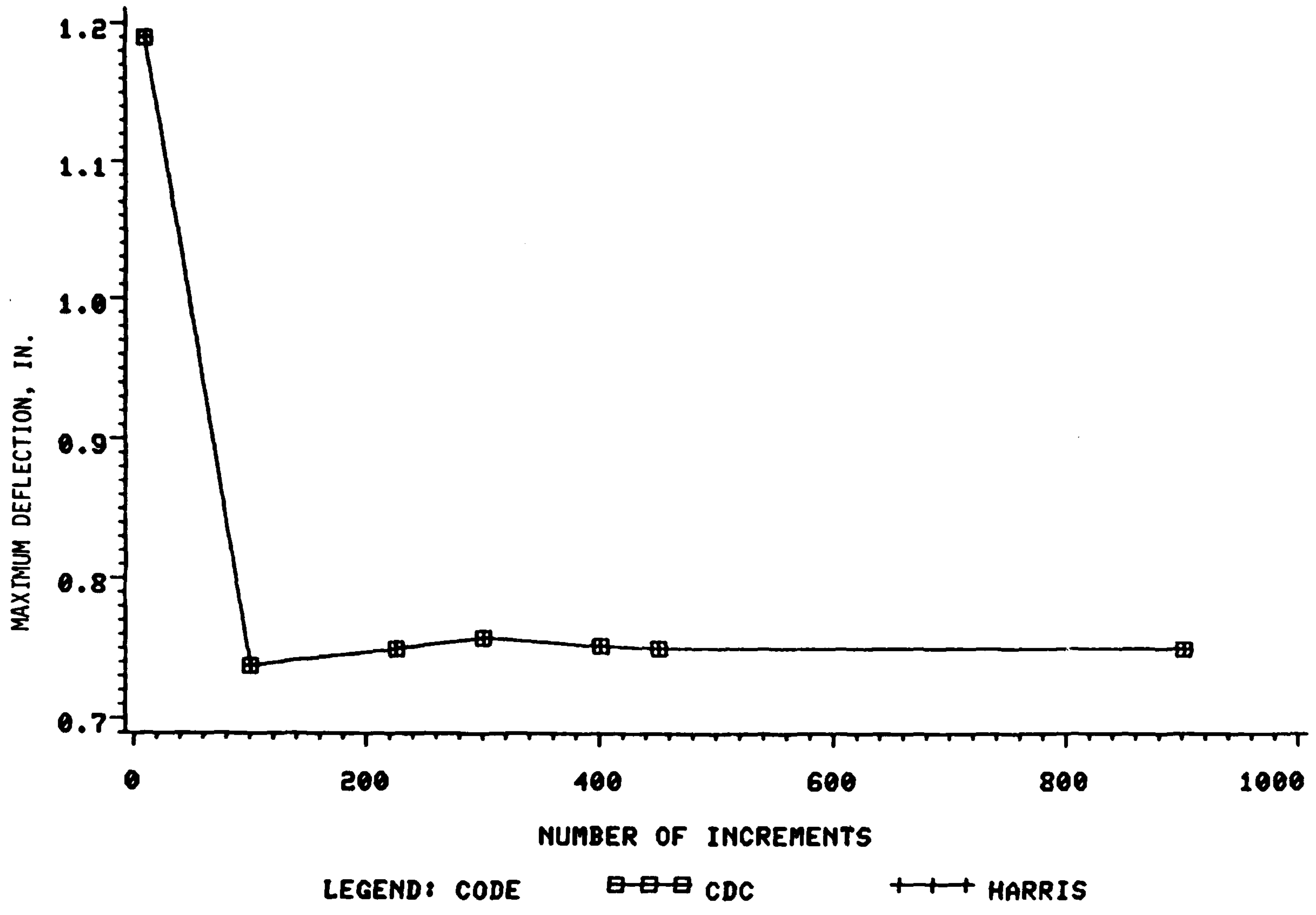


Figure 15. Maximum deflection versus number of increments, Case 1, quadruple-precision

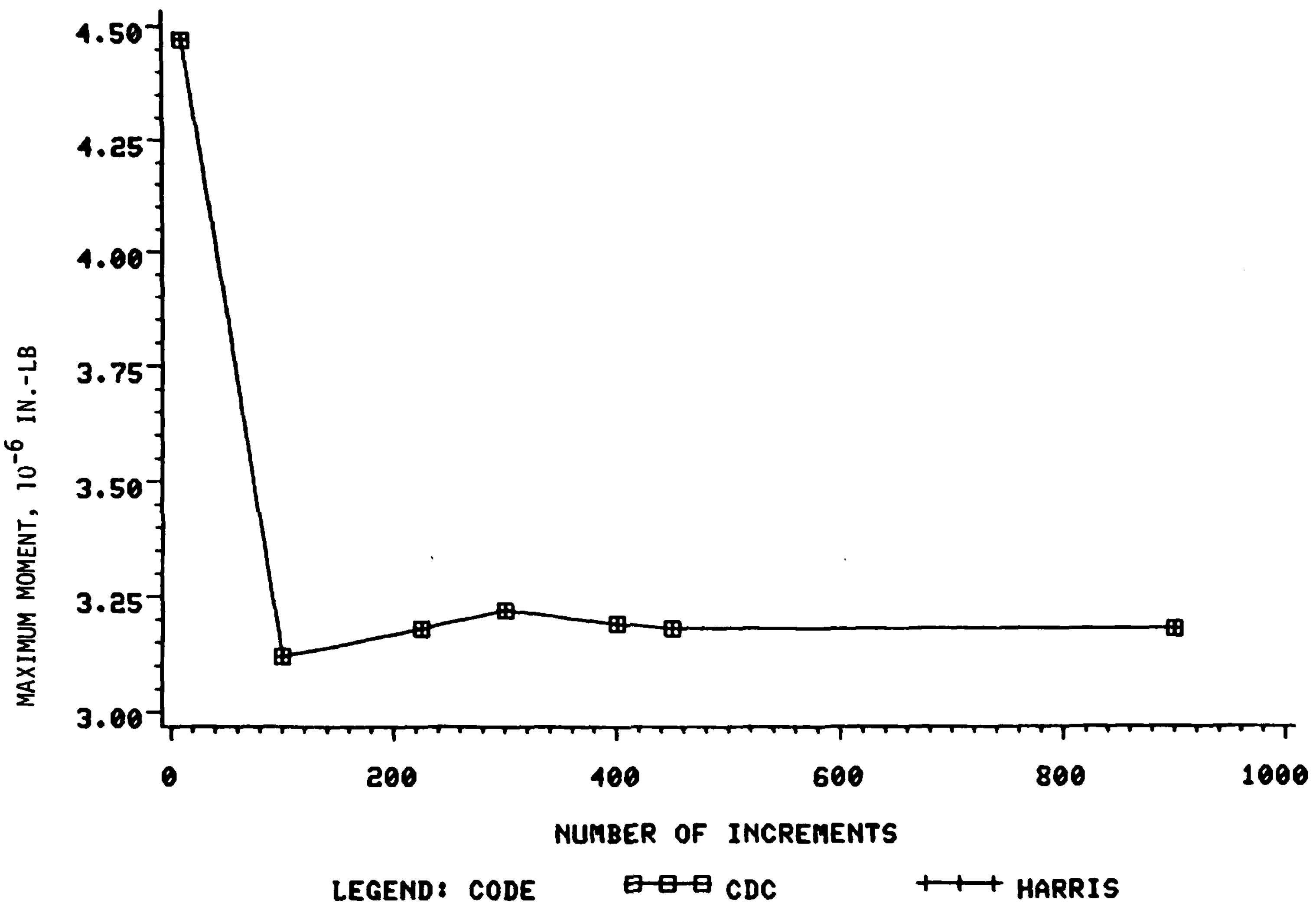


Figure 16. Maximum moment versus number of increments, Case 1, quadruple-precision

slightly only on the 900-increment run. Thus, it can be concluded that the only acceptable system for single-precision execution of Case 1 is the Harris, while all systems are acceptable in double-precision.

Case 2--steel pile in clay

68. A steel pipe pile is driven into soft clay, as shown in Figure 17. A load of 31.4 kips* is applied to the butt of the pile, resulting in an applied load of 5000 lb/radian as viewed from the top. It is desired to determine the distribution of total vertical stress in the soil surrounding the pile.

69. The problem was solved using a simple axisymmetric finite element computer code called AXSYM (Wilson 1965), assuming linear stress-strain behavior in both the pile material and the soil. The elastic parameters are shown in Figure 17, and the finite element model is depicted in Figure 18. The particular code used in this study employs quadrilateral elements composed of four constant strain triangles. As with most axisymmetric finite element codes, two stiffness equations are developed for each node in the system (one for vertical and one for horizontal displacement), and the equations are assembled into a global matrix equation. The pairs of equations containing stiffness contributions from the pile (e.g., for node 13) contain terms of large magnitude, whereas those containing stiffness contributions only from the soil (e.g., node 14) contain terms of much smaller magnitude. The ratio of these magnitudes is very roughly the ratio of the elastic moduli of the materials. Equation pairs appear in the global equation in order of nodal numbering; hence, since nodes are numbered horizontally to minimize matrix bandwidth, along each horizontal row of nodes there exists a pair of rows in the stiffness matrix with large terms followed by a pair with very small terms.

70. The global stiffness matrix equation is solved by a decomposition procedure that is mathematically equivalent to Gaussian elimination. In essence, the terms in the rows of the matrix having large-magnitude entries are multiplied by ratios obtained by dividing small terms in the row being reduced by the large-magnitude entries. These results are then added to the small terms in the row being reduced. The effect is that, except for the terms being eliminated, the small terms in the row being reduced are changed

* A table of factors for converting non-SI units of measurement to SI (metric) units is presented on page 3.

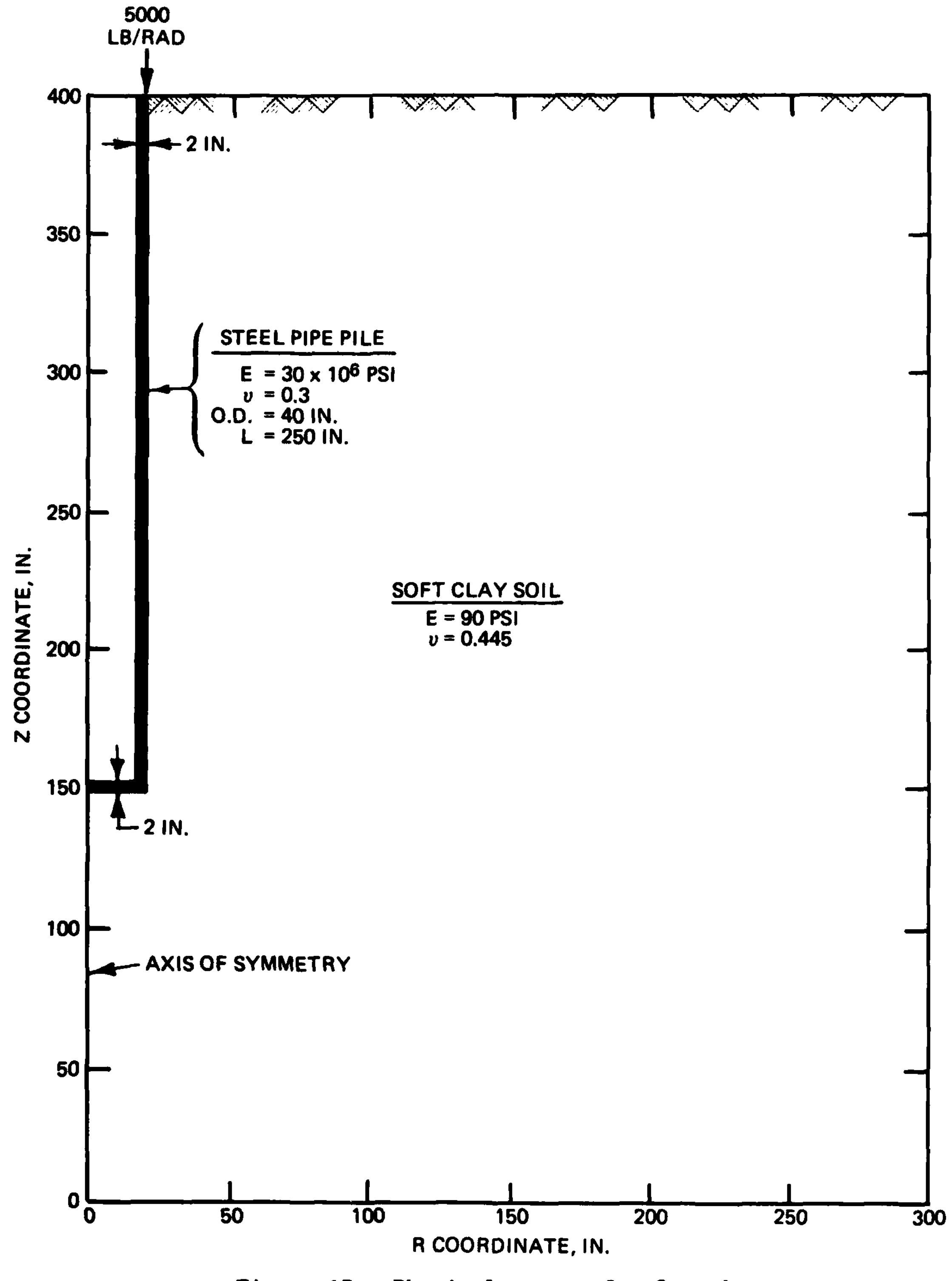


Figure 17. Physical system for Case 2

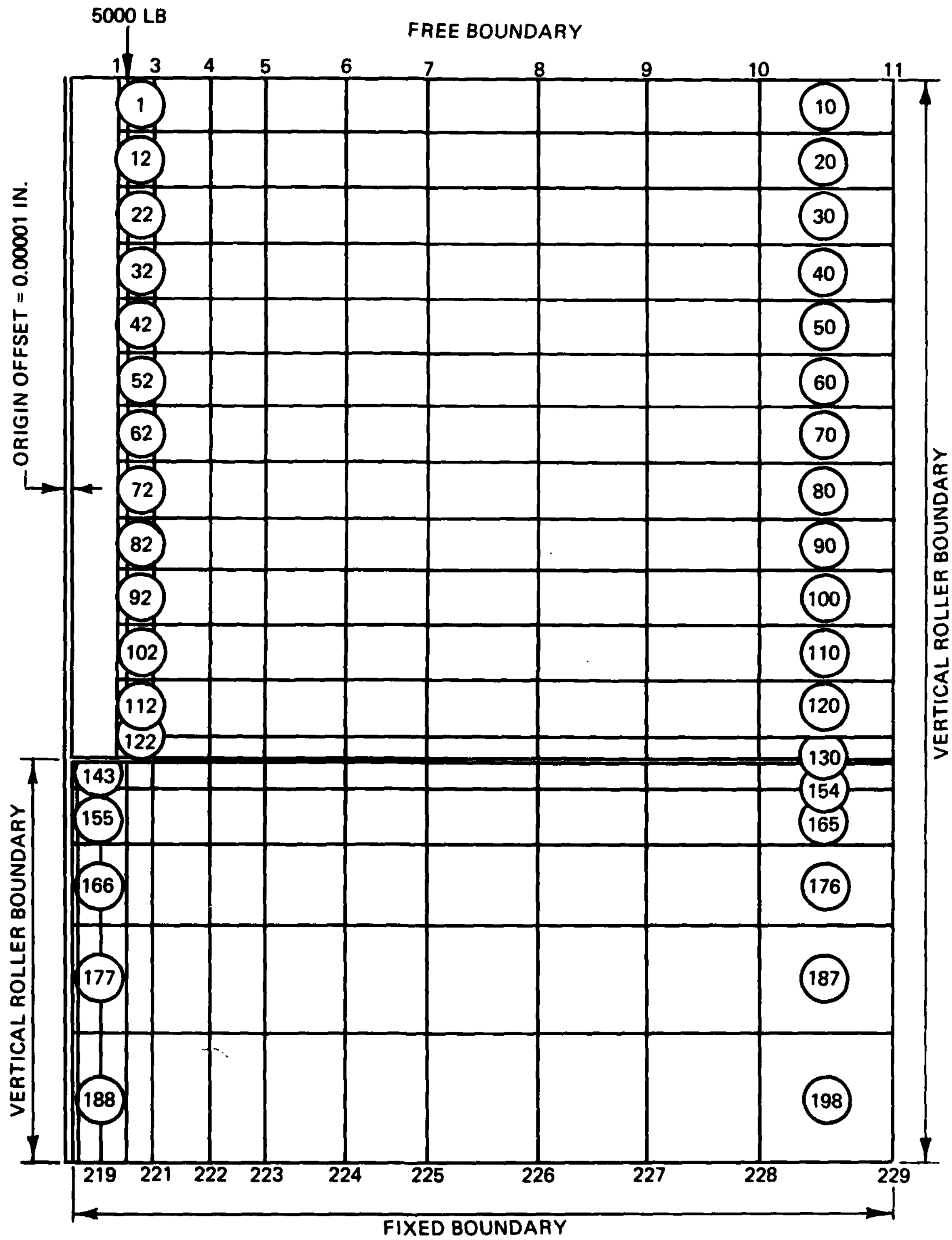


Figure 18. Finite element model for Case 2

only in their higher significant figures, and some of the significance of the physical system may be lost if the number of significant figures being used in the calculations is insufficient. Since many such "hard-soft" interfaces occur, the errors so produced become mangified.

71. Errors of the type just described can be reduced (but not eliminated) by restructuring the matrix stiffness equation (renumbering the nodes), but restructuring can result in matrices with larger bandwidths requiring considerably more storage.

72. The solutions to the problem are summarized in Figures 19-21. Figure 19 shows that the Harris single-precision solution is the only one that matches the CDC solution. The PRIME double-precision solution (Figure 20) varies from the CDC solution when the depth is less than 50 in. All other double- and quadruple-precision runs match those from the CDC.

Case 3--pile buckling analysis

73. A problem of interest to the geotechnical engineer is the computation of the buckling load on a pile driven through soft soil to bedrock. The physical problem depicted in Figure 22 was solved using a version of the BMCOL finite difference computer code described in Case 1 that includes axial load in the formulation of the equations at the nodes. However, in this case, it was necessary to model load-deflection characteristics of the soil in a nonlinear fashion. This so-called "p-y" input, shown for a typical increment of the pile in Figure 22, is based on published criteria for one-time loading of the soil. Other necessary data are described in Figure 22.

74. The applied load was assumed to act with an eccentricity of 1 in., resulting in a concentric axial load and a moment at the top of the pile. The load was increased from the Euler buckling load of approximately 300 kips (assuming the pile is pinned at both ends) to approximately 900 kips. The moment was also increased in proportion to the loads. No axial load transfer was assumed to occur between the top and the tip. Primary output needed to evaluate the buckling load is lateral deflection at the top of the pile versus applied load. Figures 23-35 show the solutions obtained on each of the systems. The Harris results conformed closely to the CDC results except that substantial differences occurred at or near the buckling point (Figures 23-25). The other systems were badly in error in single-precision but were identical with the CDC solution in double-precision. The Harris quadruple-precision result was also identical with the CDC solution.

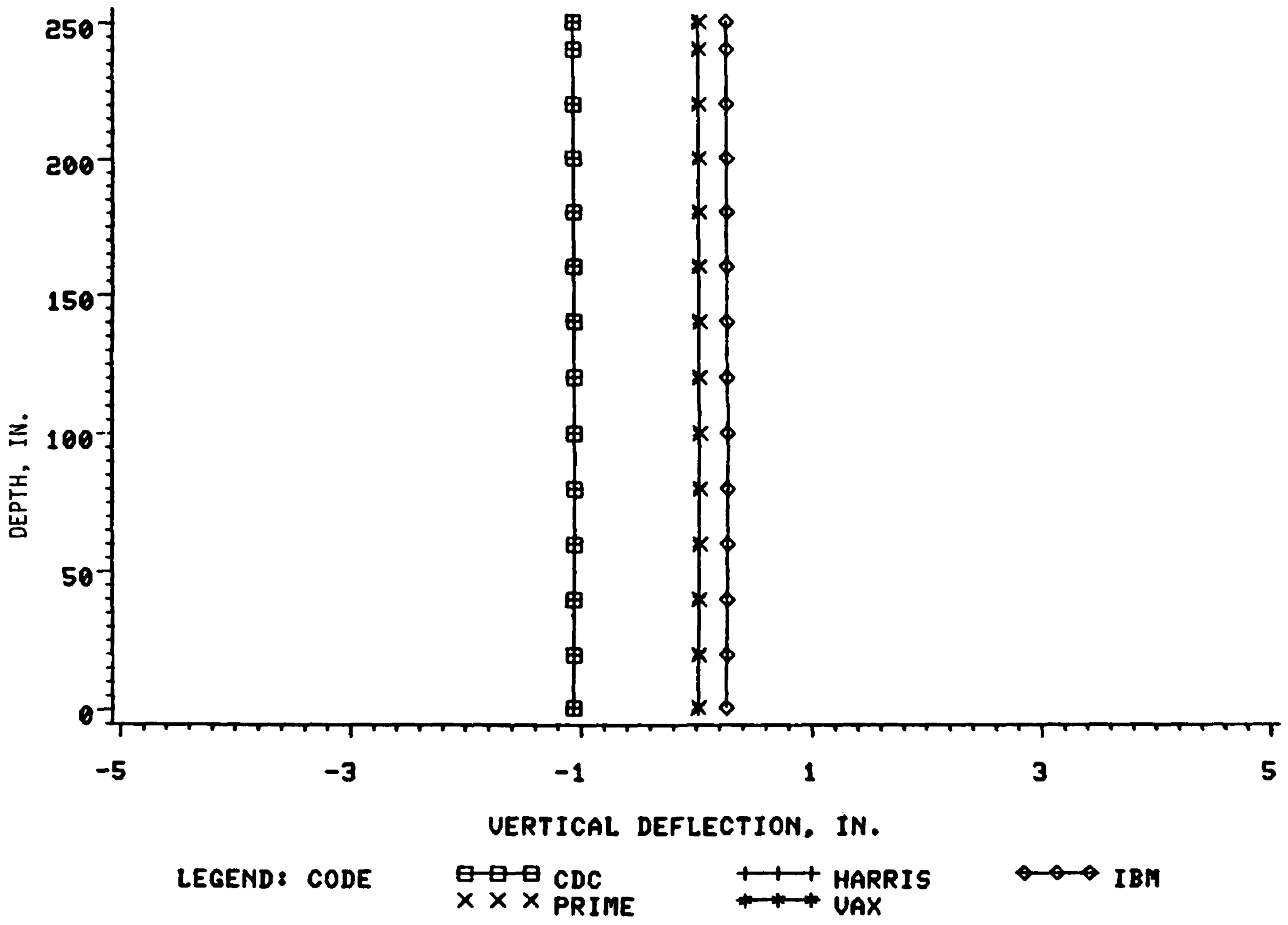


Figure 19. Vertical deflection along pile, Case 2, single-precision

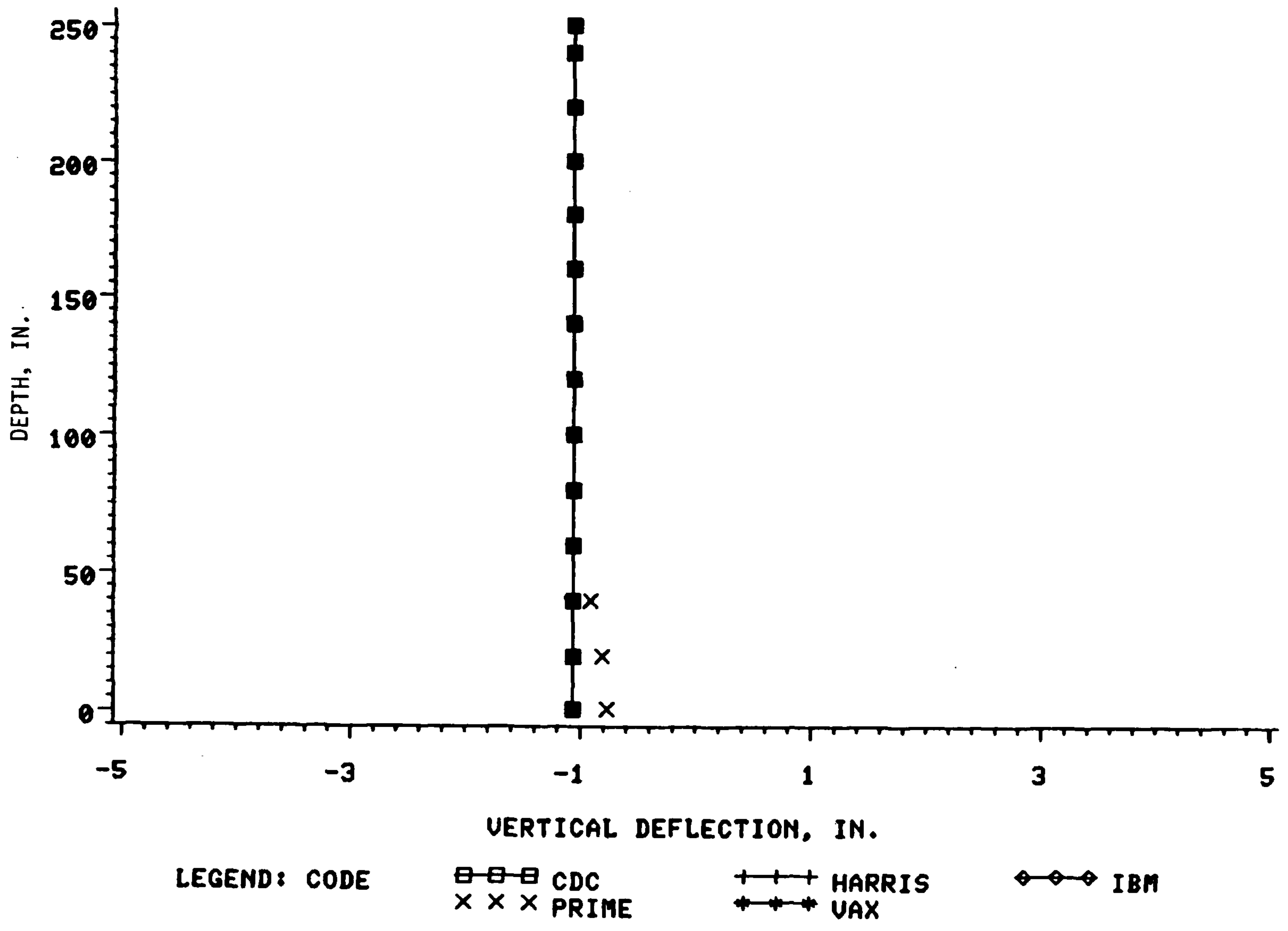


Figure 20. Vertical deflection along pile, Case 2, double-precision

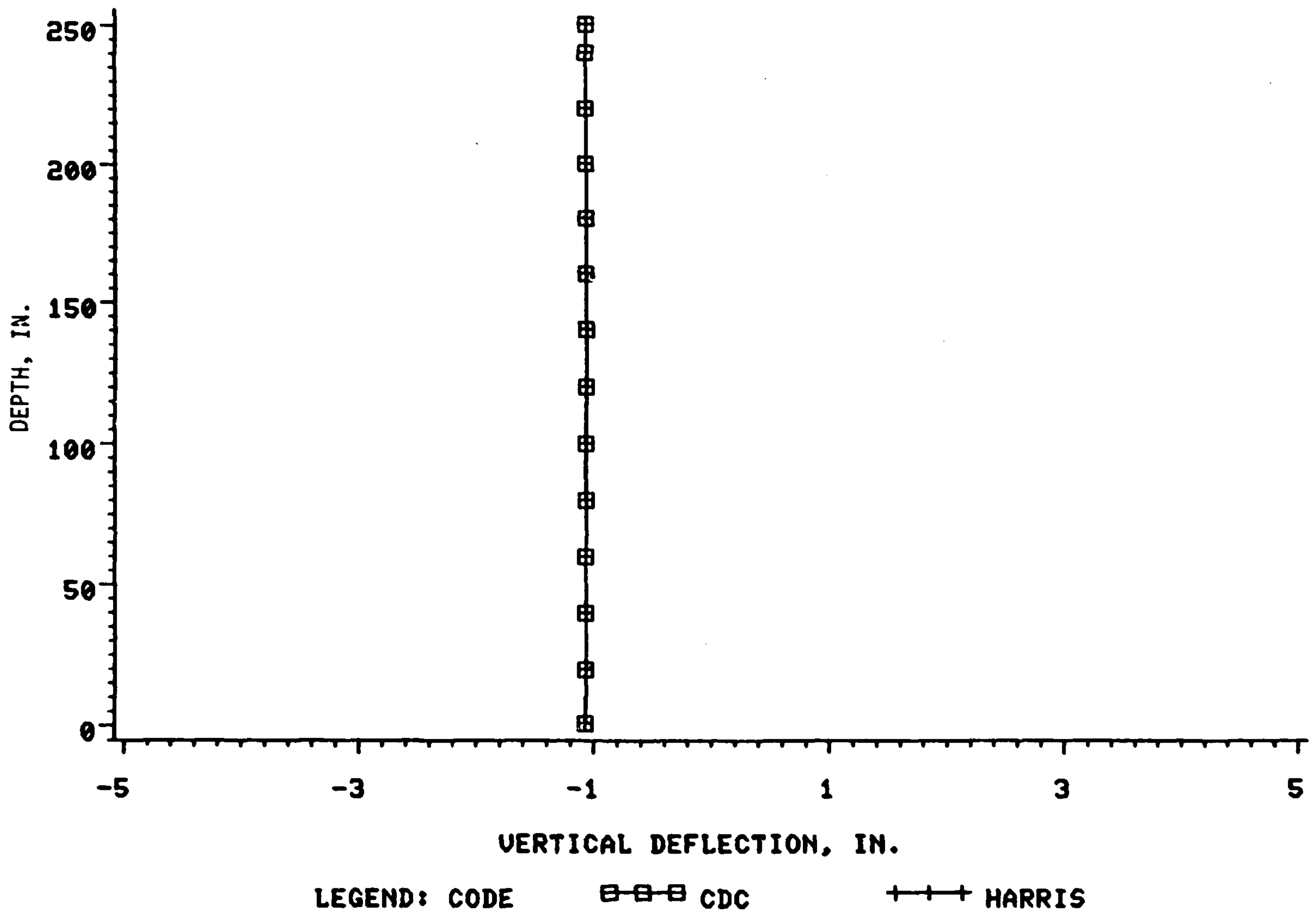
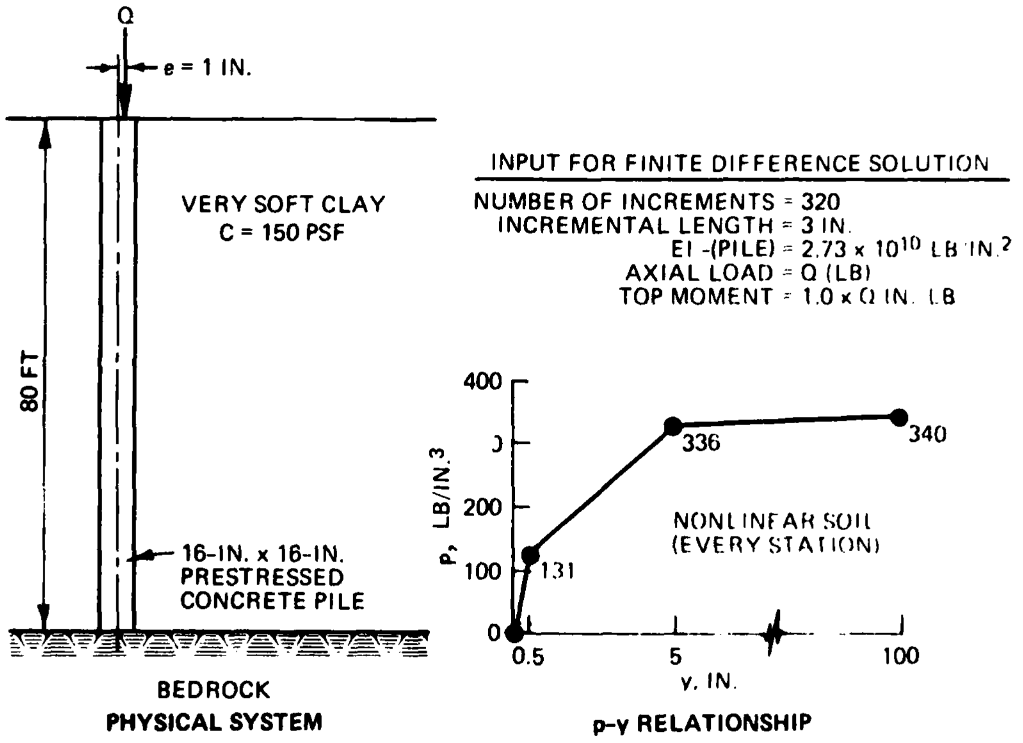


Figure 21. Vertical deflection along pile, Case 2, quadruple-precision



INPUT FOR FINITE DIFFERENCE SOLUTION

NUMBER OF INCREMENTS = 320
INCREMENTAL LENGTH = 3 IN.
EI -(PILE) = 2.73×10^{10} LB IN.^{2}
AXIAL LOAD = Q (LB)
TOP MOMENT = 1.0 x Q IN. LB

Figure 22. Physical system and finite difference input for Case 3

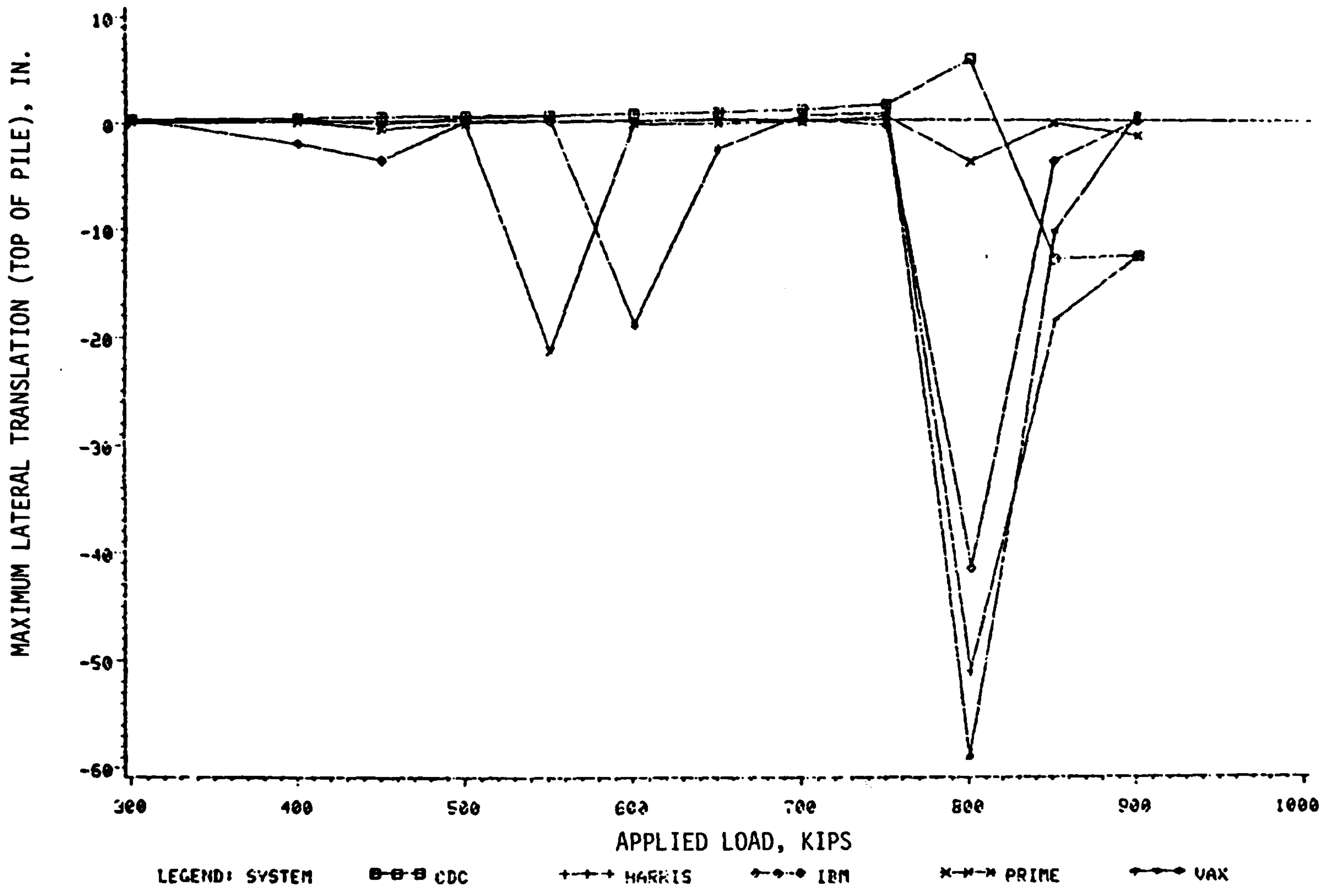


Figure 23. Maximum lateral deflection versus applied load, Case 3, single-precision, all systems

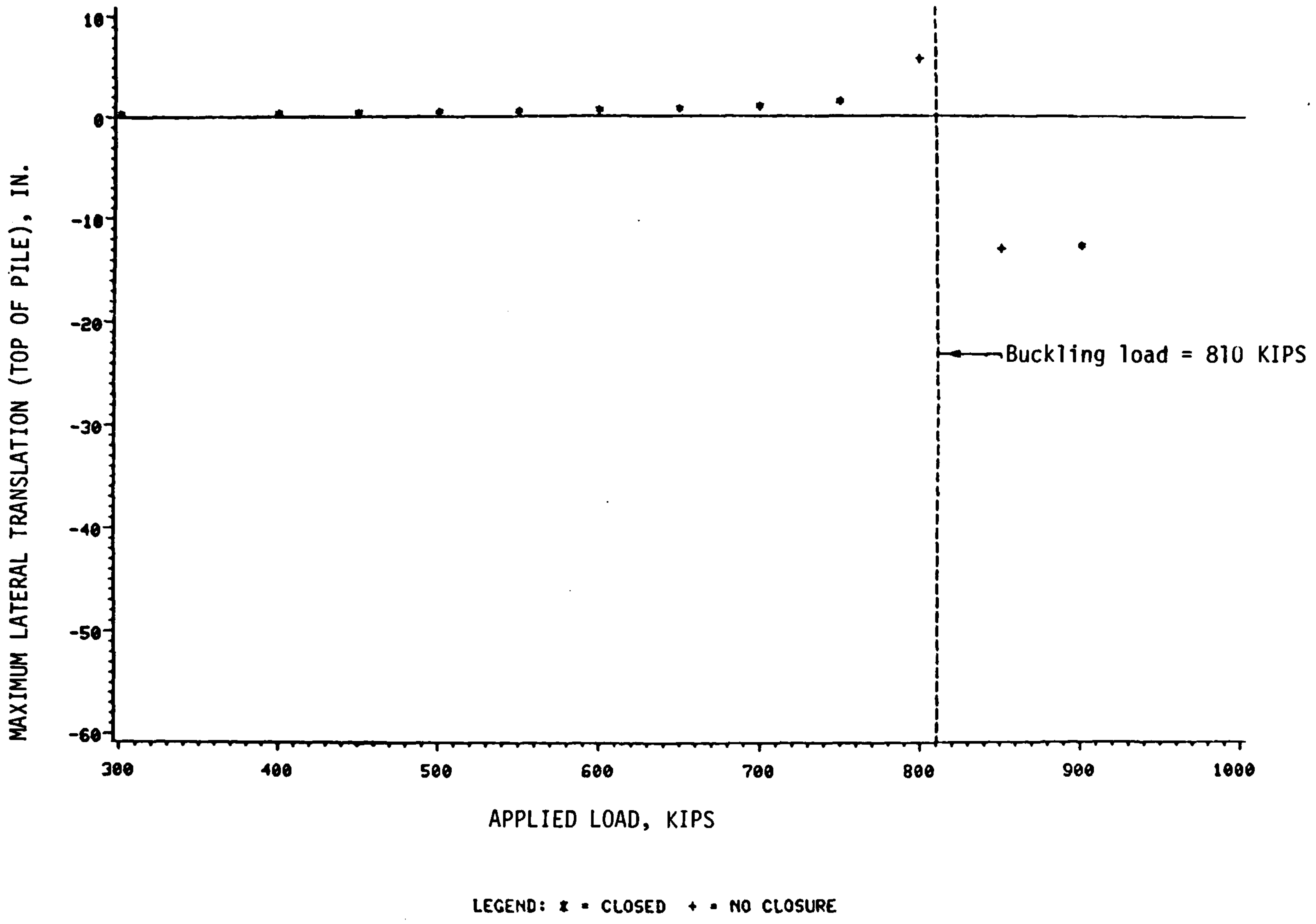


Figure 24. Maximum lateral deflection versus applied load, Case 3, single-precision, CDC system

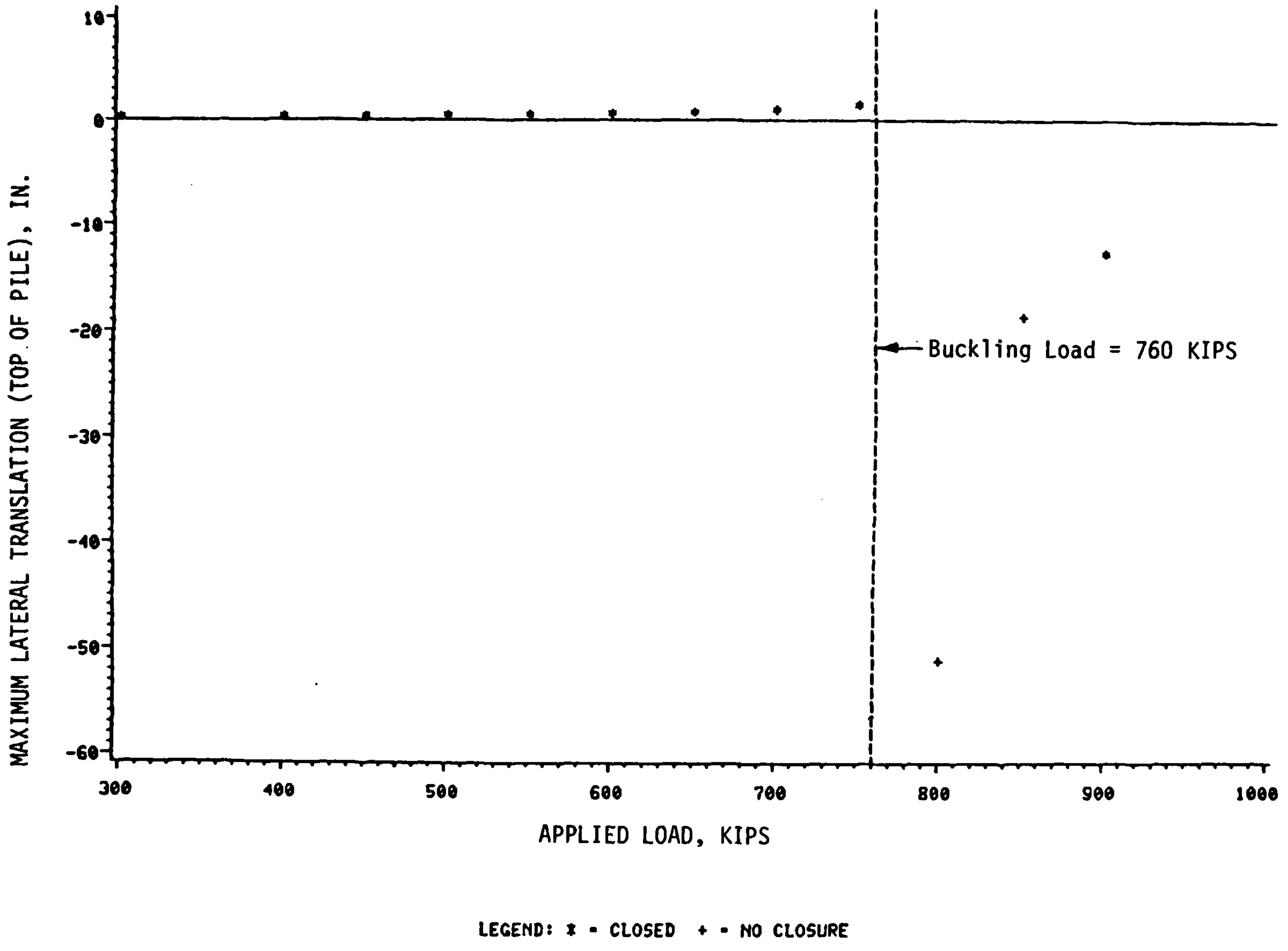


Figure 25. Maximum lateral deflection versus applied load, Case 3, single-precision, Harris system

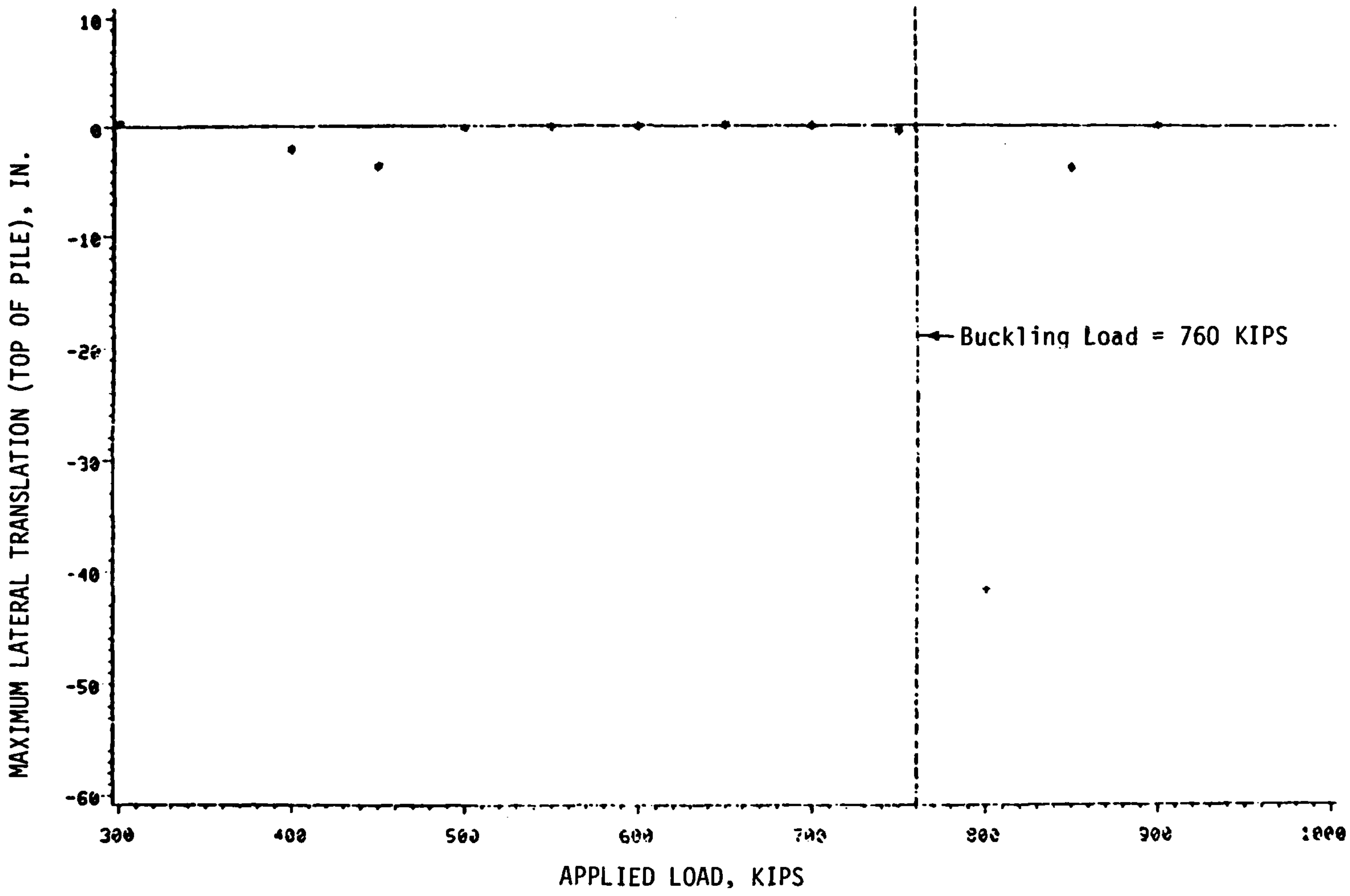


Figure 26. Maximum lateral deflection versus applied load, Case 3, single-precision, IBM system

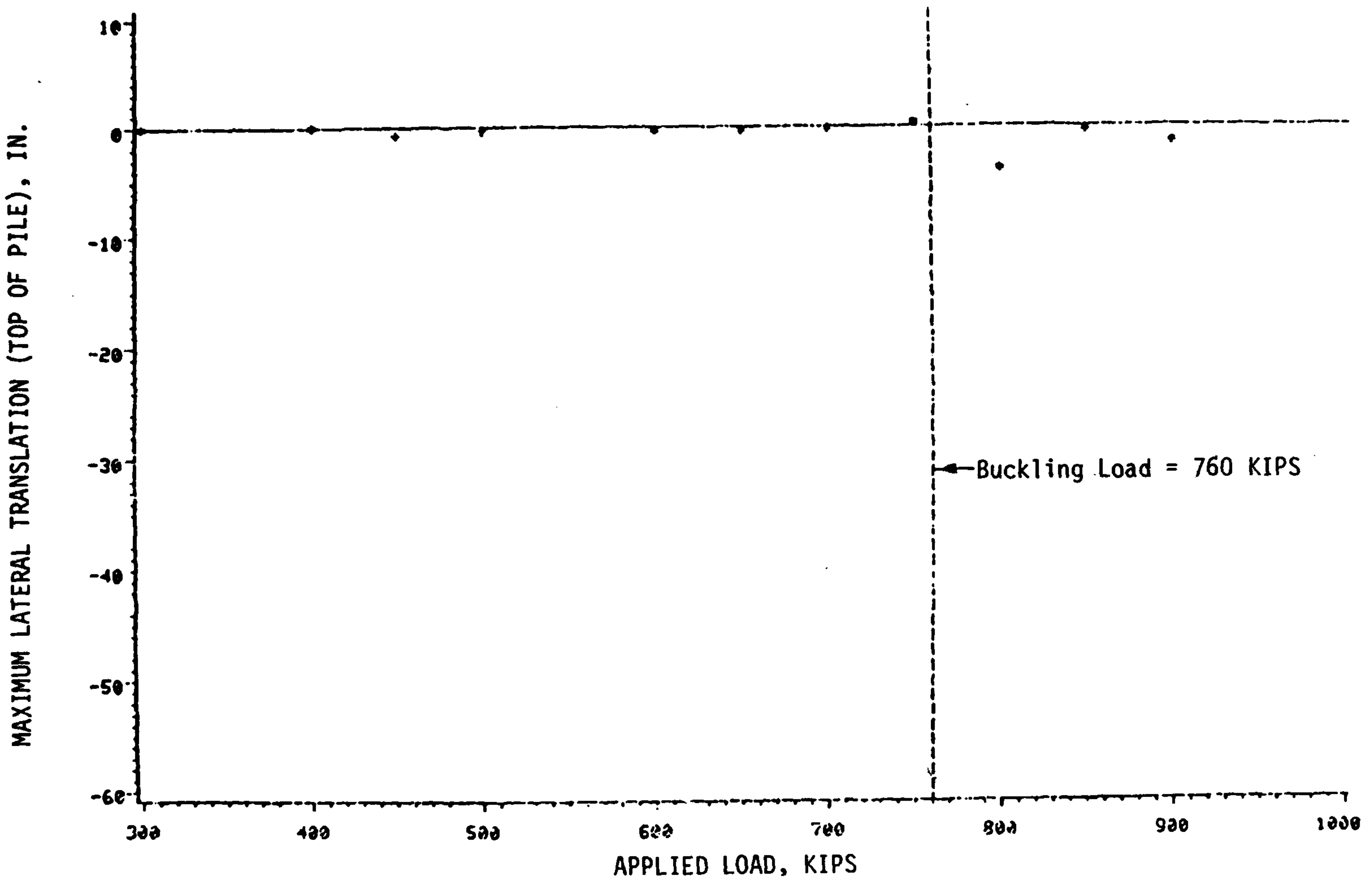


Figure 27. Maximum lateral deflection versus applied load, Case 3, single-precision, PRIME system

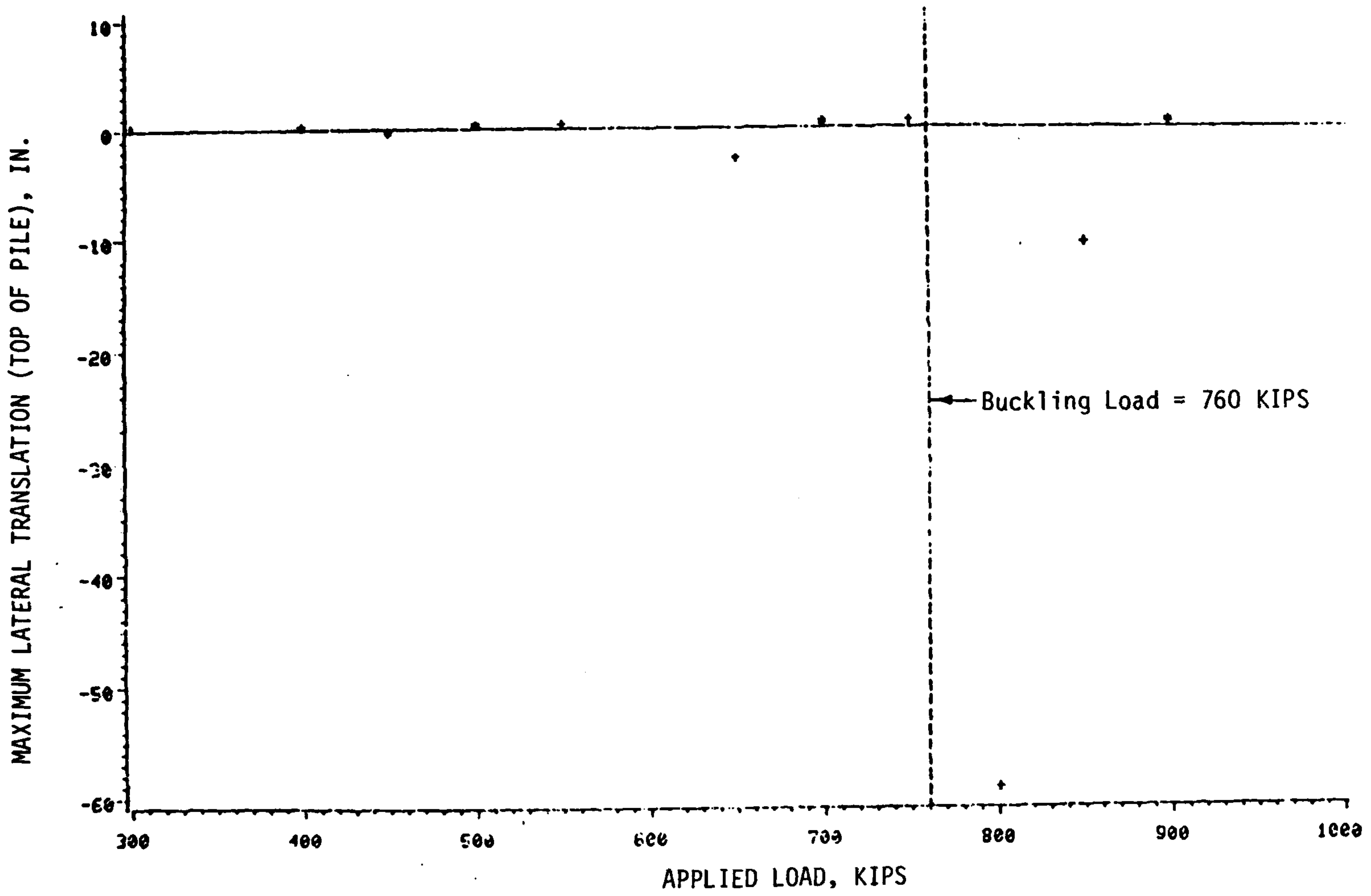


Figure 28. Maximum lateral deflection versus applied load, Case 3, single-precision, VAX system

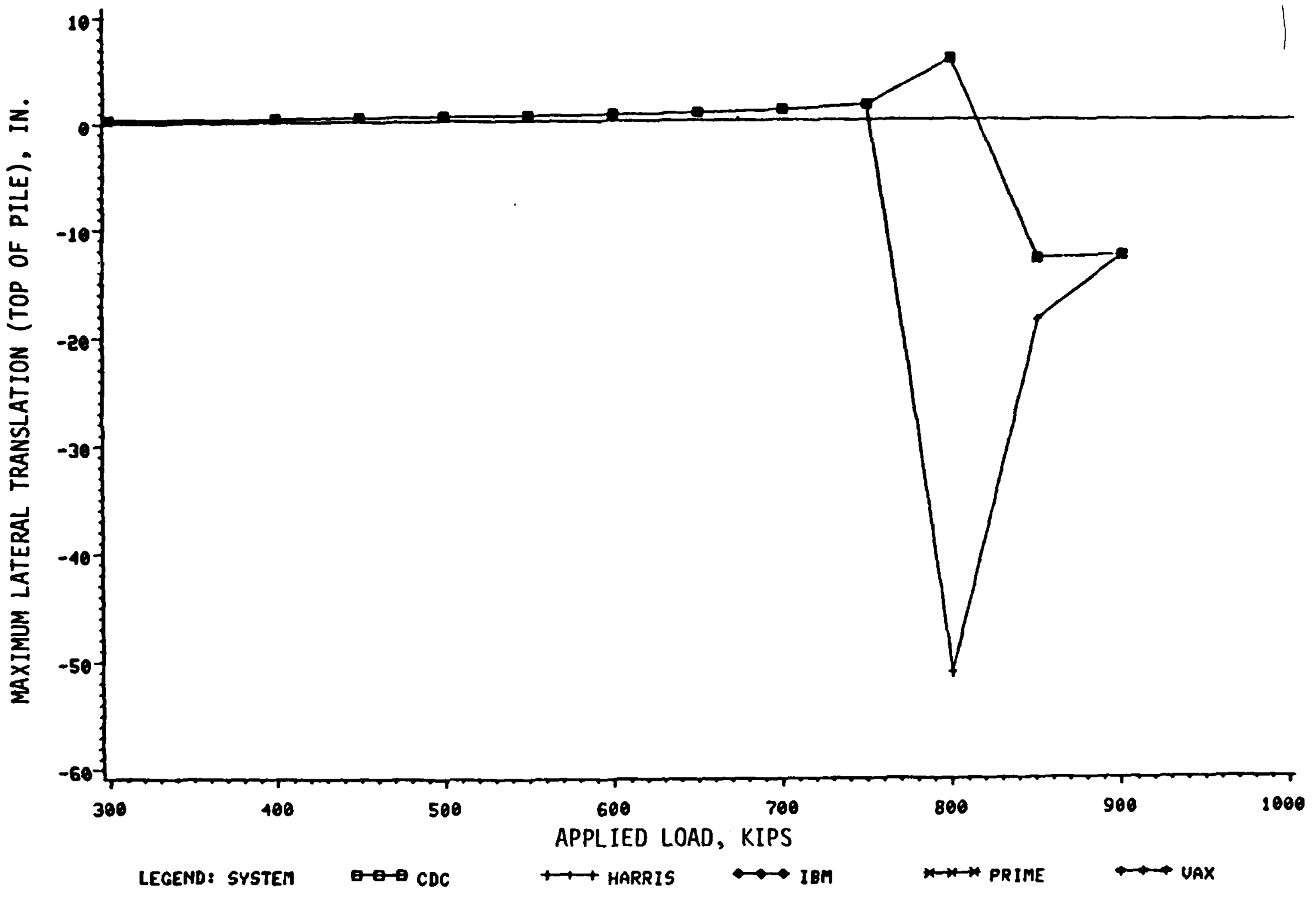


Figure 29. Maximum lateral deflection versus applied load, Case 3, double-precision, all systems

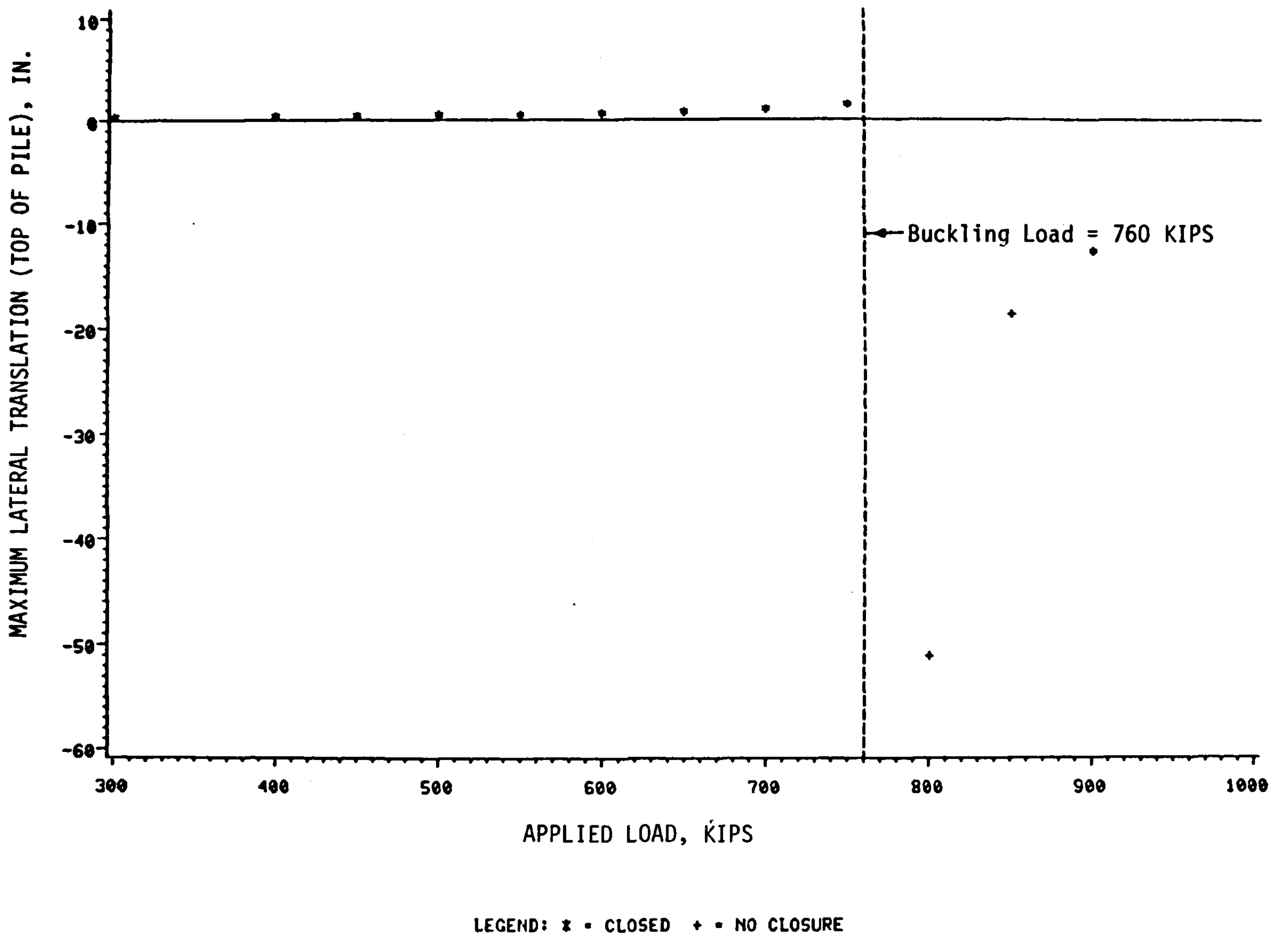


Figure 30. Maximum lateral deflection versus applied load, Case 3, double-precision, Harris system

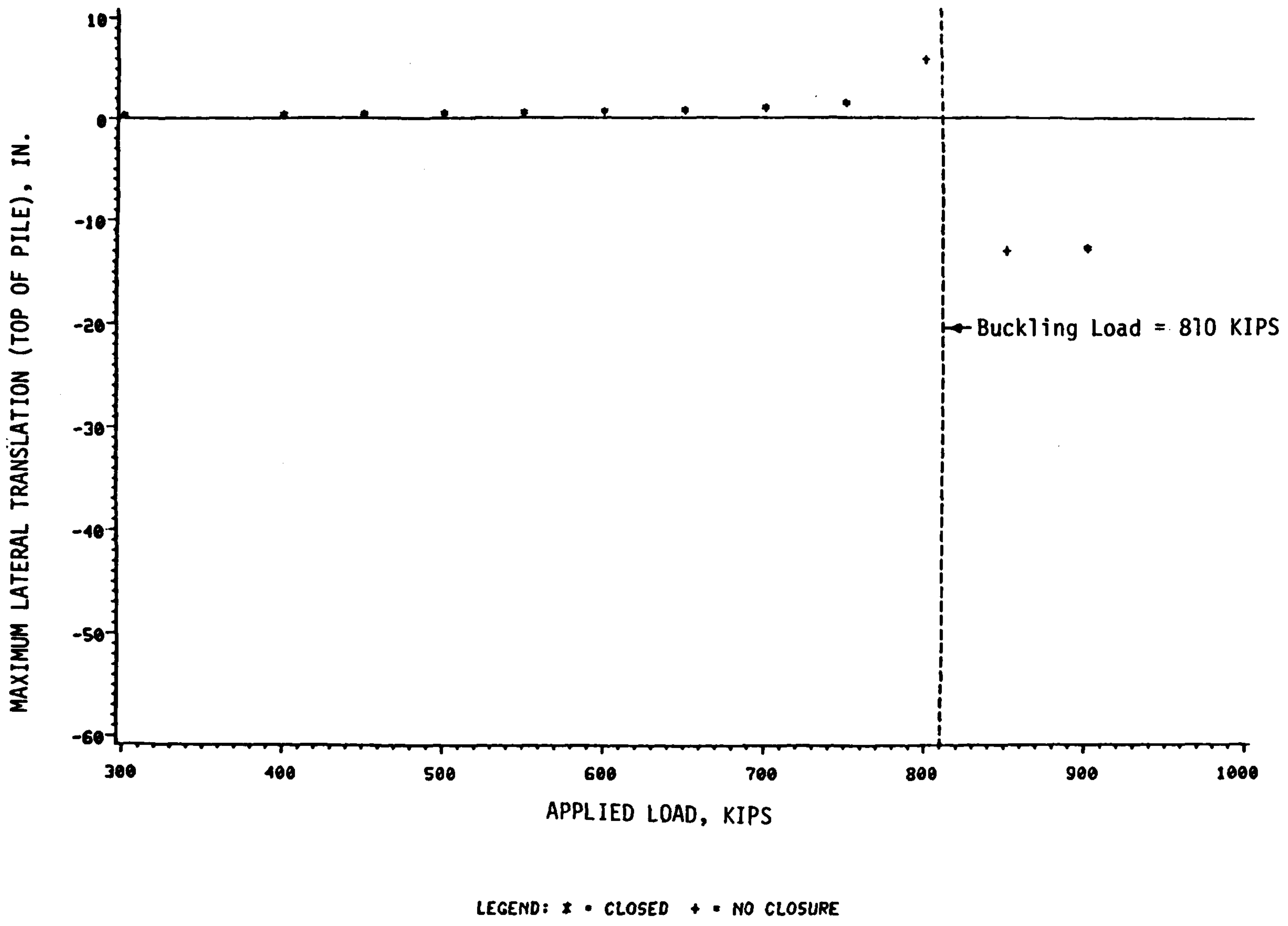


Figure 31. Maximum lateral deflection versus applied load, Case 3, double-precision, IBM system

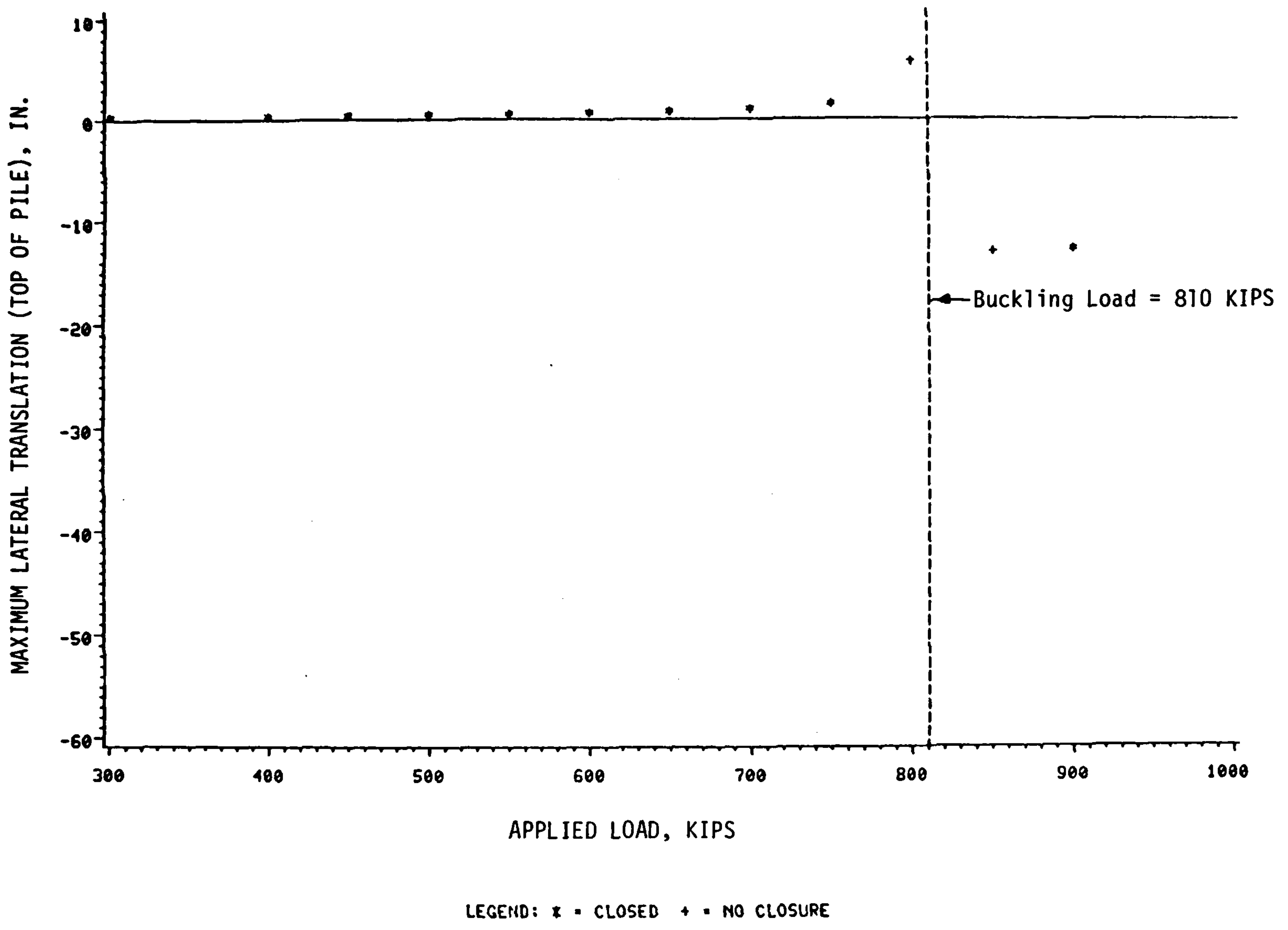


Figure 32. Maximum lateral deflection versus applied load, Case 3, double-precision, PRIME system

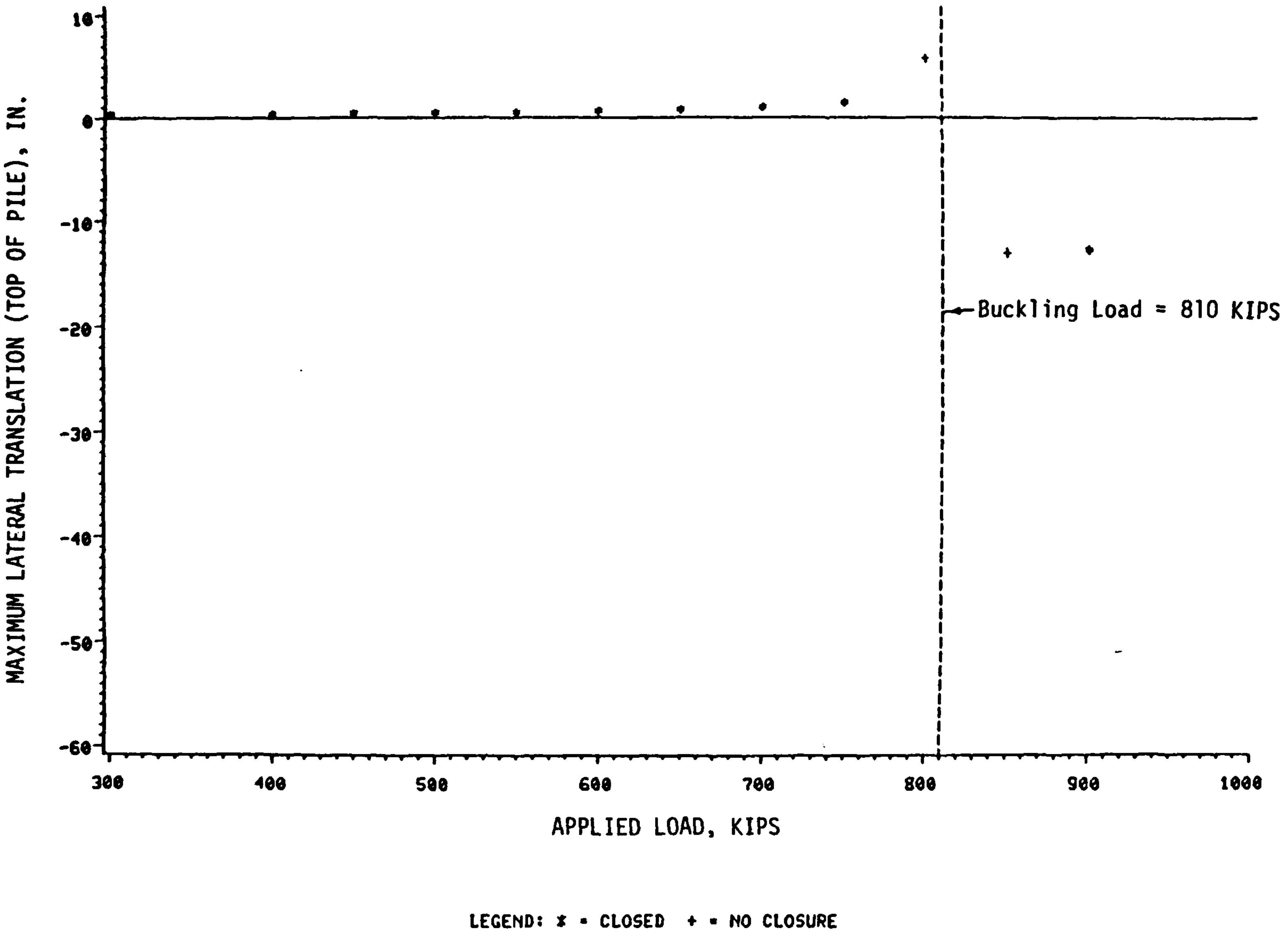


Figure 33. Maximum lateral deflection versus applied load, Case 3, double-precision, VAX system

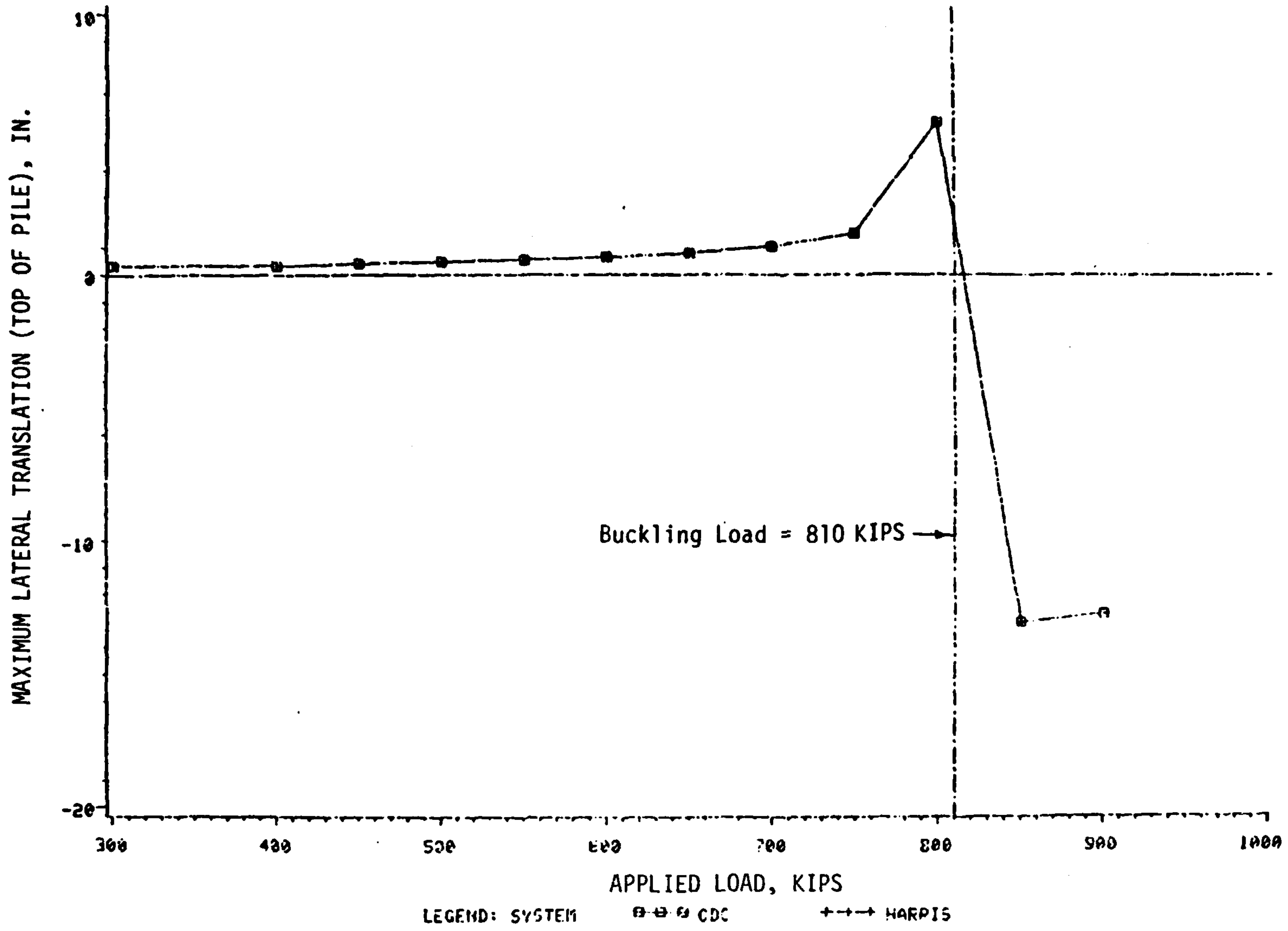


Figure 34. Maximum lateral deflection versus applied load, Case 3, quadruple-precision, CDC and Harris systems

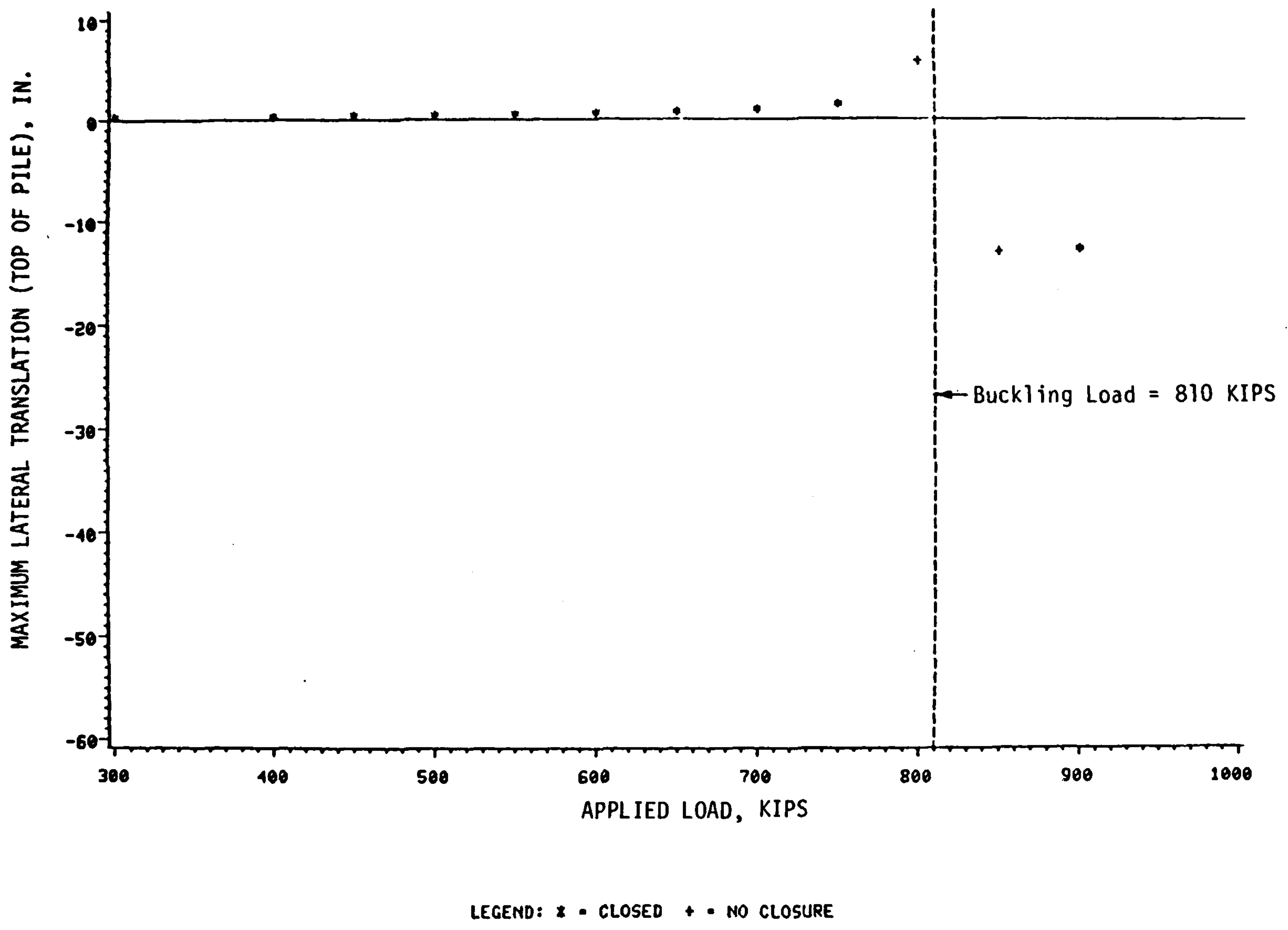


Figure 35. Maximum lateral deflection versus applied load, Case 3, quadruple-precision, Harris system

PART VI: EXECUTION TIME COMPARISONS

75. While it was not a primary objective of this study to compare test program execution times on the systems tested, accuracy considerations do interrelate with execution efficiency since there is often a trade-off between accuracy achieved and execution time expended. In fact, increased accuracy is always possible through software implementation of extended-precision arithmetic which exacts a heavy toll in execution time. And, as stated earlier, execution efficiency of FP operations cannot be ignored when evaluating minicomputer systems for S&E applications. Thus, the compile and execution times which were collected as a by-product of running the test programs are presented in Tables 6-9. Some points to note are:

a. As would be expected, compile time differences between single-precision and double-precision compilation on a given machine were in each case insignificant.

b. For most programs, compile times on the three primary systems tested show that the VAX was fastest, followed by the PRIME, and then the Harris, with time ratios (slowest to fastest) of no more than approximately 1.5. There were exceptions to this ranking; e.g., the PRIME compile time for program MATRIX did not fit the pattern.

c. Execution time differences between single-precision and double-precision object codes on a given machine were in most cases insignificant. There was no clear pattern to indicate that double-precision execution required substantially more time than single-precision on any of the three systems.

d. For the larger, longer running "real world" programs, the execution time rankings were the same as those for compile time; i.e., VAX, PRIME, and Harris, in that order, with time ratios (slowest to fastest) ranging from approximately 1.5 to 2.6. However, for the shorter "textbook" programs, the speed rankings were Harris, VAX, then PRIME, with the Harris far outperforming the other two in these cases. Since the Harris had the slowest times for the "real world" programs and the fastest times for the shorter, "textbook" programs, it might be surmized that the textbook program execution times are probably dominated by program initialization time, with the Harris outperforming the other two systems in this task. Regardless of the actual explanation of the contradictory timings, the real-world execution times are probably much more valuable as performance comparison metrics.

e. The PRIME, VAX, and IBM execution times in double-precision were generally shorter than those observed for the Harris in single- or double-precision.

Table 6

Compile Time Comparisons by Program

System	Compile Time, sec, for Cited Precision Operation		
	Single	Double	Quadruple
		Program MAXJ	
CDC	0.031		
Harris	0.96	0.98	1.00
IBM	1.85	1.49	
PRIME	1.027	1.000	
VAX	0.98	0.97	
		Program POLY	
CDC	0.093		
Harris	2.66	2.68	2.81
IBM	2.46	2.42	
PRIME	2.448	2.481	
VAX	2.01	2.07	
		Program MATRIX	
CDC	0.089		
Harris	3.00	2.37	2.49
IBM	2.24	2.37	
PRIME	0.124	0.166	
VAX	1.76	1.81	

Table 7

Compile Time Comparisons by Case

System	Compile Time, sec, for Cited Precision Operation		
	Single	Double	Quadruple
		Case 1	
CDC	0.954		
Harris	25.87	25.69	28.18
IBM	29.35	30.26	
PRIME	19.008	21.475	
VAX	16.24	15.89	
		Case 2	
CDC	1.578		
Harris	44.96	44.59	45.2
IBM	19.95	20.80	
PRIME	35.067	38.227	
VAX	26.52	26.51	
		Case 3	
CDC	0.954		
Harris	25.87	25.69	28.18
IBM	29.35	30.26	
PRIME	19.008	21.475	
VAX	16.30	16.42	

Table 8

Execution Time Comparisons by Program

System	Execution Time, sec, for Cited Precision Operation		
	Single	Double	Quadruple
	Program POLY		
CDC	0.046		
Harris	0.581	0.706	1.98
IBM	2.62	3.17	
PRIME	1.218	1.306	
VAX	0.95	1.08	
	Program MATRIX		
CDC	0.019		
Harris	0.204	0.204	0.206
IBM	1.08	1.10	
PRIME	0.568	0.557	
VAX	0.43	0.45	

Table 9

Execution Time Comparisons by Case

System	Execution Time, sec, for Cited Precision Operation		
	Single	Double	Quadruple
	Case 1		
CDC	1.417		
Harris	19.89	18.97	49.54
IBM	82.21	130.34	
PRIME	17.036	19.075	
VAX	12.29	13.48	
	Case 2		
CDC	3.629		
Harris	58.04	58.10	254.58
IBM	83.15	139.51	
PRIME	34.112	35.190	
VAX	22.68	31.66	
	Case 3		
CDC	3.04		
Harris	37.73	38.54	181.80
IBM	53.34	90.31	
PRIME	36.001	37.848	
VAX	25.42	27.26	

f. Harris quadruple-precision execution times, except for one textbook program (MATRIX), were substantially longer than Harris double-precision times. For real world programs, quadruple/double time ratios ranged as high as approximately 5.0. For programs with a high percentage of total execution time expenditure in FP computations, it is certainly conceivable that the ratio may go substantially higher. Such time penalties are generally unacceptable for production execution of long-running S&E programs.

76. The timing comparisons are presented here only as interesting observations that came from the accuracy tests. The reader should avoid drawing erroneous conclusions. Specifically:

a. No "benchmark" or performance study was intended or designed. Programs were selected only to measure computational accuracy and not to measure performance. The program mix and indeed the entire approach to the problem would have been substantially different had this been a comparative performance study.

b. Past experience has shown that program timings obtained on some minicomputer systems are greatly affected by system load conditions; on some systems much more than others. For example, an identical program execution may register much less execution time under a "no-load" or uniprogramming condition than during a period of heavy system load. Timings presented here were not obtained under no-load conditions to eliminate system loading influences. Neither were attempts made to control load conditions during program runs so that similar conditions would exist on all systems. In fact, load conditions in most cases were unknown during program runs. Thus, the "repeatability" of the timings is highly suspect.

c. The specific systems used in this study, the VAX 11/750, the PRIME 550, and the Harris 500, were not selected to be directly competitive in performance or price. No attempt was made to determine the exact configurations on which the programs were executed, the price of the systems used, or the vendor's performance rating of the particular system and configuration relative to others in his product line or in competitor's product lines. In short, the systems were not selected as approximately equally priced alternatives for competitive procurement, but simply as three available systems with FP architectures representative of the three families of systems. Therefore, no conclusions should be drawn comparing the general performance or price/performance of the VAX 11 family versus the PRIME 550 family versus the Harris Series 500 family.

PART VII: CONCLUSIONS AND RECOMMENDATIONS

Relative Accuracy of the Three FP Architectures

77. Both examination of the FP representations and execution of the test programs indicate that, for single-precision computations, the Harris is substantially more accurate than either of the other two systems and the VAX is slightly more accurate than the PRIME. Harris superiority is achieved by use of the double-precision representation for all FP values in machines equipped with a SAU. This feature provides the equivalent of approximately 11 decimal digits. While both the VAX and the PRIME store a 23-bit fraction providing the equivalent of approximately 7 decimal digits, the VAX is more accurate due to:

a. The added "hidden" bit giving it a 24-bit rather than 23-bit fraction.

b. The achievement of correct rounding.

78. However, the test cases indicate that users should be wary of using single-precision computations for S&E problems on any of the three systems. The VAX and PRIME single-precision runs failed rather badly in all three of the "real world" test cases. While the Harris's 11-digit representation failed to provide sufficient accuracy in only one of these three cases, it too must be considered suspect since it fails to provide the level of confidence needed to allay doubts as to its accuracy when used for such problems. It must be remembered that results obtained at one or more points during computation may in fact be accurate and that, while final results may not appear unreasonable, they may in fact be wrong.

79. Double-precision representations provide approximately 16, 14, and 11 decimal digits for the VAX, PRIME, and Harris, respectively. All have hardware implementations of double-precision operations. The 16-digit VAX data format produced no failures in any of the tests and could be used with confidence for common S&E applications in the Corps requiring highly precise data values. The PRIME's 14-digit double-precision results conformed well to the CDC "standard" in all but one test case (real world Case 2), where there was a small but significant deviation. The PRIME's double-precision would probably suffice for the great majority of applications but would provide a somewhat lower level of confidence than the VAX or CDC. As previously stated

(Part III), Harris double-precision and single-precision representations are the same in SAU-equipped machines; thus, the single-precision comments above apply here as well.

80. Previous experience with software implementations of FP arithmetic leads to the conclusion that use of the Harris quadruple-precision capability would exact too great a performance penalty for many applications. Timings from the test cases executed for this study confirm this belief.

81. In summary, this study indicates that:

a. Single-precision arithmetic should be avoided as a standard practice when using either the VAX or the PRIME system for S&E applications in the Corps. This is not to say that such use should be totally prohibited, but that it should be restricted to cases where a careful analysis and thorough understanding of the problem to be solved and the programmed solution indicate that the choice is prudent. Since it is not necessarily known a priori when double-precision is needed (this may be more data-dependent than program-dependent), the cost of making a definite determination will, in most cases, be unjustified. Thus, it is recommended that double-precision be used as standard practice when using either the VAX or the PRIME system. The double-precision arithmetic of both systems proved to be highly accurate in the tests reported herein, with the VAX having a small but significant advantage.

b. Since the Harris uses the same representation for both single- and double-precision values, the options are limited to standard-precision or software-implemented quadruple-precision. Quadruple-precision execution times will probably be prohibitive for most precision-sensitive Corps applications; thus, the question reduces to whether Harris standard-precision is sufficient. Based on this study, the answer seems to be that for many (perhaps most) applications, the degree of accuracy provided is sufficient, but for some existing S&E applications in the Corps, accuracy may be a problem for the Harris system. While the Harris standard FP arithmetic certainly provides far greater computational accuracy than either the VAX or the PRIME single-precision arithmetic, it may not provide a level which will inspire confidence and allay fears in the Corps' engineer programmer community. Thus, it is recommended that, for S&E applications, all the tested systems be used with caution and that users be ever alert to indications that an application is or will be precision-sensitive. Such indications will require either an accuracy study for that application or a move to another system. Note also that it is recommended that the "P" compiler option be used as a standard practice (see Part III) so that input/output data fields are not artificially restricted to the documented single-precision length.

82. Note that blanket use of double-precision values imposes a penalty in memory required and in some systems a penalty in execution time. While

this study did not specifically address FP computational speed, no significant difference in execution time for single- versus double-precision computation was detected for any of the three systems. A further study would be required to make definitive statements regarding single- versus double-precision speed comparisons for the systems in question. Note also that any performance penalties that may accompany blanket double-precision are imposed as a default in the Harris system; i.e., every FP value requires 2 words (6 bytes). Thus, when comparison is made of resource requirements, it must be realized that the VAX or PRIME double-precision representation (8 bytes) does not require double the data memory required by Harris standard representation, but that the ratio is actually 8 bytes to 6 bytes.

Recommendations for Error Avoidance

83. Given an understanding of the basic algorithms for FP operations and the mechanics of how errors occur, and care in coding accuracy-critical calculations, errors which might otherwise be inadvertently created can be avoided. Specifically, the user should:

a. Be very careful when subtracting very nearly equal values. The result may contain very few significant digits, and loss of digits in a subsequent operation could be disastrous. Also, be aware that the distributive law may fail. (See example in Part IV.)

b. Be careful of addition or subtraction operations performed on numbers of vastly differing magnitudes. Some loss of significance is certain, and the associative law of addition can fail. (See example in Part IV.)

c. Never test for absolute equality of FP values. Instead detect approximate equality by testing for a difference in value less than some relative ε.

d. Be aware that the effectiveness of adherence to recommendations a and b can be negated by an optimizing compiler which reorganizes code assuming associativity and distributivity.

REFERENCES

Cody, W. J. and Waite, W. 1980. Software Manual for the Elementary Functions, Prentice-Hall, Englewood Cliffs, N. J.

Harris Corporation. 1978. Reference Manual, Series 500 General Purpose Digital Computer Systems, Fort Lauderdale, Fla.

________. 1981. Harris FORTRAN Reference Manual, Fort Lauderdale, Fla.

Knuth, D. E. 1969. The Art of Computer Programming, Vol 2, Addison-Wesley, Reading, Mass.

Matlock, H. and Ingram, W. B. 1963. "Bending and Buckling of Soil-Supported Structural Elements," Proceedings, Second Pan American Conference on Soil Mechanics and Foundation Engineering, São Paulo, Brazil.

Noble, B. 1982. "Accuracy Comparison Study," Letter Report, University of Wisconsin-Madison, Madison, Wis.

O'Neill, M. W. and Peterson, E. H. 1976. "Analysis of Machine Dependent Errors in Soil-Structure Interaction," Department of Civil Engineering, University of Houston, Houston, Tex.

Ward, D. L. 1976. "The Impact of Word Length in Scientific Computation," unpublished paper, Texas Eastern University, Tyler, Tex.

Wilson, E. L. 1965. "Structural Analysis of Axisymmetric Solids," AIAA Journal, Vol 3, No. 12, New York.

BIBLIOGRAPHY

Digital Equipment Corporation. 1979. VAX 11 Architecture Handbook, Maynard, Mass.

__________. 1980. VAX Technical Summary, Maynard, Mass.

Katzan, H., Jr. 1977. Microprogramming Primer, McGraw-Hill, New York.

Prime Computer, Inc. 1981. Assembly Language Programmer's Guide, Framingham, Mass.

__________. 1981. System Architecture Reference Guide, Framingham, Mass.

Ralston, A. 1965. A First Course in Numerical Analysis, McGraw-Hill, New York.

Shugan, T. A. 1982. "Relative Accuracy of the Computed Matrix Inverse on Small and Large Computers," Technical Memorandum No. M-51-82-06, Naval Civil Engineering Laboratory, Port Hueneme, Calif.

Stone, H. S. 1975. Introduction to Computer Architecture, SRA, Inc., Chicago, Ill.

APPENDIX A: RESULTS OF MACHAR, AN ENVIRONMENTAL INQUIRY SUBROUTINE

1. Subroutine MACHAR (Cody and Waite 1980) dynamically determines 13 machine constants relating to the floating-point arithmetic system. These constants can be used to check manufacturer's claims or documentation for a system and are specified below:

IBETA	- Base of the floating-point representation
IT	- Number of base IBETA digits in the floating-point significand
IRND	- 0 if floating-point addition truncates 1 if floating-point addition rounds
NGRD	- Number of guard digits for multiplication: 0 if IRND = 1, or if IRND = 0 and only IT base IBETA digits participate in the post-normalization shift of the floating-point significand in multiplication 1 if IRND = 0 and more than IT base IBETA digits participate in the post-normalization shift of the floating-point significand in multiplication
MACHEP	- Largest negative integer such that 1.0+FLOAT(IBETA)**MACHEP.NE.1.0, except that MACHEP is bounded below by -(IT+3)
NEGEPS	- Largest negative integer such that 1.0-FLOAT(IBETA)**NEGEPS.NE.1.0, except that NEGEPS is bounded below by -(IT+3)
IEXP	- Number of bits (decimal places if IBETA = 10), reserved for the representation of the exponent (including the bias or sign) of a floating-point number
MINEXP	- Largest magnitude negative integer such that FLOAT(IBETA)**MINEXP is a positive floating-point number
MAXEXP	- Largest positive integer exponent for a finite floating-point number
EPS	- Smallest positive floating-point number such that 1.0+EPS.NE.1.0. In particular, if either IBETA = 2 or IRND = 0, EPS = FLOAT(IBETA)**MACHEP. Otherwise, EPS = (FLOAT(IBETA)**MACHEP)/2
EPSNEG	- A small positive floating-point number such that 1.0-EPSNEG.NE.1.0. In particular, if IBETA = 2 or IRND = 0, EPSNEG = FLOAT(IBETA)**NEGEPS. Otherwise, EPSNEG = (IBETA**NEGEPS)/2. Because NEGEPS is bounded below by -(IT+3), EPSNEG may not be the smallest number which can alter 1.0 by subtraction
XMIN	- Smallest nonvanishing floating-point power of the radix. In particular, XMIN = FLOAT(IBETA)**MINEXP

XMAX - Largest finite floating-point number. In particular, XMAX = (1.0-EPSNEG)*FLOAT(IBETA)**MAXEXP. Note: On some machines, XMAX will be only the second, or perhaps third, largest number, being too small by 1 or 2 units in the last digit of the significand

2. These constants are discussed further in Cody and Waite (1980). The results of MACHAR are given in Table A1. Subroutine MACHAR is listed on pages A4-A6.

Table A1

Values of 13 Machine Constants Determined Using Subroutine MACHAR

Constant	Value of Constant Determined on Cited System				
	CDC	Harris	IBM	PRIME	VAX
IBETA	2	2	16	2	2
IT-Single	48	38	24	23	24
IT-Double	NR*	38	56	47	56
IT-Quad	N/A	69	N/A	N/A	N/A
IRND	1	0	0	0	1
NGRD	0	1	1	1	0
MACHEP-Single	-48	-37	-5	-46**	-24
MACHEP-Double	NR	-37	-15	-46	-56
MACHEP-Quad	N/A	-68	N/A	N/A	N/A
NEGEPS-Single	-47	-37	-6	-46**	-24
NEGEPS-Double	NR	-37	-14	-46	-56
NEGEPS-Quad	N/A	-68	N/A	N/A	N/A
IEXP-Single	†	0†	0†	0†	0†
IEXP-Double	NR	0†	0†	1†	0†
IEXP-Quad	N/A	0†	N/A	N/A	N/A
MINEXP-Single	-975	-128	-65	-129	-128
MINEXP-Double	NR	-128	-65	-32896	-128
MINEXP-Quad	N/A	-8388607	N/A	N/A	N/A
MAXEXP-Single	1070	127	63	127	127
MAXEXP-Double	NR	127	63	98176	127
MAXEXP-Quad	N/A	8388607	N/A	N/A	N/A
EPS-Single	.35527127 E-14	.72757561 E-11	.95367432 E-6	.14210855 E-13**	.59604645 E-7
EPS-Double	NR	.72759561 E-11	.22055605 E-15	.14693681 E-38†	.13877288 E-16
EPS-Quad	N/A	.338813179 E-20	N/A	N/A	N/A
EPSNEG-Single	.71054274 E-14	.72759561 E-11	.59604645 E-7	.14210885 E-13**	.59604645 E-7
EPSNEG-Double	NR	.72759561 E-11	.13877788 E-16	.14210855 E-13	.13877288 E-16
EPSNEG-Quad	N/A	.338813179 E-20	N/A	N/A	N/A
XMIN-Single	.313151306 E-293	.293873588 E-38	.539760535 E-78	.1469367794 E-38	.293873588 E-38
XMIN-Double	NR	.293873588 E-38	.539760535 E-78	††	.293873588 E-38
XMIN-Quad	N/A	1.17247385 E-2525223	N/A	N/A	N/A
XMAX-Single	.126501408 E+323	.170141183 E+39	.723700588 E+76	.170141163 E+39	.170141173 E+39
XMAX-Double	NR	.170141183 E+39	.723700588 E+76	††	.170141173 E+39
XMAX-Quad	N/A	2.13224371 E+252522	N/A	N/A	N/A

* NR--not run.

** This value appears to reflect the effect of MACHAR computations being performed entirely within the PRIME system's arithmetic unit; i.e., without incurring the truncation of bits that accompanies the storing of an FP value to memory. Therefore, direct comparison of this PRIME constant to the corresponding constant from the other systems may not be relevant in a general sense.

† MACHAR produces obviously erroneous values for this constant.

†† MACHAR produces an FP overflow while attempting to compute this constant.

```
      INTEGER I,IBETA,IEXP,IRND,IT,IZ,J,K,MACHEP,
     &      MX,NEGEP,NGRD
      DOUBLE PRECISION A,B,BETA,BETAIN,BETAM1,EPS,EPSNEG,ONE,XMAX,
     &             XMIN,Y,Z,ZERO
C
C      THIS ROUTINE IS INTENDED TO DETERMINE THE CHARACTERISTICS OF
C      THE FLOATING-POINT ARITHMETIC SYSTEM THAT ARE SPECIFIED BELOW.
C
C      IBETA  - THE RADIX OF THE FLOATING-POINT REPRESENTION
C      IT     - THE NUMBER OF BASE IBETA DIGITS IN THE FLOATING-POINT
C               SIGNIFICAND
C      IRND   - 0 IF FLOATING-POINT ADDITION CHOPS,
C               1 IF FLOATING-POINT ADDITION ROUNDS
C      NGRD   - THE NUMBER OF GUARD DIGITS FOR MULTIPLICATION.  IT IS
C               0 IF IRND=1, OR IF IRND=0 AND ONLY IT BASE. IBETA
C                 DIGITS PARTICIPATE IN THE POST NORMALIZATION SHIFT
C                 OF THE FLOATING-POINT SIGNIFICAND IN MULTIPLICATION
C               1 IF IRND=0 AND MORE THAN IT BASE IBETA DIGITS
C                 PARTICIPATE IN THE POST NORMILIZATION SHIFT OF THE
C                 FLOATING-POINT SIGNIFICAND IN MULTIPLICATION
C      MACHEP - THE LARGEST NEGATIVE INTEGER SUCH THAT
C               1.0+FLOAT(IBETA)**MACHEP .NE. 1.0, EXCEPT THAT
C               MACHEP IS BOUNDED BELOW BY -(IT+3)
C      NEGEPS - THE LARGEST NEGATIVE INTEGER SUCH THAT
C               1.0-FLOAT(IBETA)**NEGEPS .NE. 1.0, EXCEPT THAT
C               NEGEPS IS BOUNDED BELOW BY -(IT+3)
C      IEXP   - THE NUMBER OF BITS (DECIMAL PLACES IF IBETA = 10)
C               RESERVED FOR THE REPRESENTATION OF THE EXPONENT
C               (INCLUDING THE BIAS OR SIGN) OF A FLOATING-POINT
C               NUMBER
C      MINEXP - THE LARGEST IN MAGNITUDE NEGATIVE INTEGER SUCH THAT
C               FLOAT(IBETA)**MINEXP IS A POSITIVE FLOATING-POINT
C               NUMBER
C      MAXEXP - THE LARGEST POSITIVE INTEGER EXPONENT FOR A FINITE
C               FLOATING-POINT NUMBER
C      ESP    - THE SMALLEST POSITIVE FLOATING-POINT NUMBER SUCH
C               THAT 1.0+EPS .NE. 1.0. IN PARTICULAR, IF EITHER
C               IBETA = 2 OR IRND = 0, EPS = FLOAT(IBETA)**MACHEP.
C                OTHERWISE.  EPS = (FLOA(IBETA)**MACHEP)/2
C      EPSNEG - A SMALL POSITIVE FLOATING-POINT NUMBER SUCH THAT
C               1.0-EPSNEG .NE. 1.0 IN PARTICULAR.  IF IBETA = 2
C               OR IRND=0, EPSNEG = FLOAT(IBETA)**NEGEPS.
C               OTHERWISE, EPSNEG = (IBETA**NEGEPS)/2. BECAUSE
C               NEGEPS IS BOUNDED BELOW BY -(IT+3). EPSNEG MAY NOT
C               BE THE SMALLEST NUMBER WHICH CAN ALTER 1.0 BY
C               SUBTRACTION.
C      XMIN   - THE SMALLEST NON-VANISHING FLOATING-POINT POWER OF THE
C               RADIX. IN PARTICULAR, XMIN = FLOAT(IBETA)**MINEXP
C      XMAX   - THE LARGEST FINITE FLOATING-POINT NUMBER. IN
C               PARTICULAR XMAX = (1.0-EPSNEG)*FLOAT(IBETA)**MAXEXP
C               NOTE - ON SOME MACHINES XMAX WILL BE ONLY THE
C               SECOND, OR PERHAPS THIRD, LARGEST NUMBER, BEING
C               TOO SMALL BY 1 OR 2 UNITS IN THE LAST DIGIT OF
C               THE SIGNIFICAND.
C
C     LATEST REVISION - OCTOBER 22, 1979
C
C      AUTHOR - W. J. CODY
C               ARGONNE NATIONAL LABORATORY
C
C-------------------------------------------------------------------
      CALL MACHAR(IBETA,IT,IRND,NGRD,MACHEP,NEGEP,IEXP,MINEXP,
     &MAXEXP,EPS,EPSNEG,XMIN,XMAX)
C
      WRITE(6,10)IBETA,IT,IRND,NGRD,MACHEP,NEGEP,IEXP,MINEXP
     &           ,MAXEXP,EPS,EPSNEG,XMIN,XMAX
   10 FORMAT(1X,'IBETA  = ',I10, /,
     &       1X,'IT     = ',I10, /,
     &       1X,'IRND   = ',I10, /,
     &       1X,'NGRD   = ',I10, /,
     &       1X,'MACHEP = ',I10, /,
     &       1X,'NEGEPS = ',I10, /,
     &       1X,'IEXP   = ',I10, /,
     &       1X,'MINEXP = ',I10, /,
     &       1X,'MAXEXP = ',I10, /,
     &       1X,'EPX    = ',D15.9, /,
     &       1X,'EPSNEG = ',D15.9, /,
     &       1X,'XMIN   = ',D15.9, /,
     &       1X,'XMAX   = ',D15.9)
      STOP
      END
      SUBROUTINE MACHAR(IBETA, IT, IRND, NGRD, MACHEP, NEGEP,
     &                  IEXP, MINEXP, MAXEXP, EPS, EPSNEG,XMIN,
     &                         XMAX)
C
      INTEGER I,IBETA, IEXP,IRND, IT, IZ, J, K, MACHEP,
     &       MAXEXP, MINEXP, MX, NEGEP, NGRD
      DOUBLE PRECISION A,B,BETA,BETAIN,BETAM1,
     &      ESP,EPSNEG,ONE,XMAX,XMIN,Y,Z,ZERO
C
C      THIS SUBROUTINE IS INTENDED TO DETERMINE THE
C      CHARACTERISTICS OF THE FLOATING-POINT ARITHMETIC
C      SYSTEM THAT ARE SPECIFIED BELOW.
C
C-----------------------------------------------------------
      ONE=DBLE(FLOAT(1))
      ZERO = 0.0D0
C--------------------------------------------------------------
C     DETERMINE IBETA,BETA ALA MALCOLM
C--------------------------------------------------------------
      A = ONE
   10 A = A + A
        IF (((A+ONE)-A)-ONE .EQ. ZERO) GO TO 10
      B = ONE
   20 B = B + B
         IF((A+B)-A .EQ. ZERO) GO TO 20
      IBETA = INT(SNGL((A + B) - A))
      BETA = DBLE(FLOAT(IBETA))
C------------------------------------------------------------------
C     DETERMINE IT, IRND
      IT = 0
      B = ONE
  100 IT = IT + 1
      B = B * BETA
      IF(((B + ONE)-B)-ONE .EQ. ZERO) GO TO 100
      IRND = 0
      BETAM1 = BETA -ONE
      IF((A+BETAM1)-A .NE. ZERO)IRND = 1
C------------------------------------------------------------------
C      DETERMINE NEGEP, ESPSNEG
C------------------------------------------------------------------
      NEGEP = IT + 3
      BETAIN = ONE/BETA
      A = ONE
C
      DO 200 I = 1,NEGEP
      A = A * BETAIN
  200 CONTINUE
C
      B = A
  210 IF((ONE-A)-ONE .NE.ZERO)GO TO 220
      A = A * BETA
      NEGEP = NEGEP - 1
      GO TO 210
  220 NEGEP = -NEGEP
      EPSNEG = A
      IF((IBETA .EQ. 2) .OR. (IRND .EQ. 0)) GO TO 300
      A = (A*(ONE+A)) /(ONE+ONE)
      IF ((ONE-A)-ONE .NE. ZERO)EPSNEG = A
C-------------------------------------------------------------------------
C     DETERMINE MACHEP, EPS
C-------------------------------------------------------------------------
  300 MACHEP = -IT - 3
      A = B
  310 IF((ONE+A)-ONE .NE. ZERO )GO TO 320
      A = A * BETA
      MACHEP = MACHEP + 1
       GO TO 310
  320 EPS = A
      IF ((IBETA .EQ. 2) .OR. (IRND .EQ. 0)) GO TO 350
      A = (A*(ONE+A))/(ONE+ONE)
      IF((ONE+A)-ONE .NE. ZERO) EPS = A
C----------------------------------------------------------------
C     DETERMINE NGRD
C----------------------------------------------------------------
  350 NGRD = 0
      IF((IRND .EQ. 0) .AND. ((ONE+EPS)*ONE-ONE) .NE. ZERO) NGRD = 1
C----------------------------------------------------------------
C      DETERMINE IEXP, MINEXP, XMIN
C
C      LOOP TO DETERMINE LARGEST I AND K = 2**1 SUCH THAT
C      (1/BETA) ** (2**(1))
C      DOES NOT UNDERFLOW
C      EXIT FORM LOOP IS SIGNALED BY AN UNDERFLOW
C------------------------------------------------------------------
      I = 0
      K = 1
      Z = BETAIN
  400 Y = Z
      Z = Y * Y
```

```
C---------------------------------------------------------------
C      CHECK FOR UNDERFLOW HERE
C---------------------------------------------------------------
      A = Z * ONE
      IF((A+A .EQ. ZERO) .OR. (DABS(Z) .GE. Y)) GO TO 410
      I = I + 1
      K = K + K
      GO TO 400
  410 IF(IBETA .EQ. 10) GO TO 420
      IEZP = I + 1
      MX = K + K
      GO TO 450
C---------------------------------------------------------------
C      FOR DECIMAL MACHINES ONLY
C---------------------------------------------------------------
  420 IEXP = 2
      IZ = IBETA
  430 IF(K.LT.IZ) GO TO 440
      IZ = IZ * IBETA
      IEXP = IEXP + 1
      GO TO 430
  440 MX = IZ + IZ - 1
C---------------------------------------------------------------
C     LOOP TO DETERMINE MINEXP, XMIN
C     EXIT FORM LOOP IS SIGNALED BY AN UNDERFLOW.
C---------------------------------------------------------------
  450 XMIN = Y
      Y = Y * BETAIN
C---------------------------------------------------------------
C      CHECK FOR UNDERFLOW HERE
C---------------------------------------------------------------
      A = Y * ONE
      IF(((A+A) .EQ. ZERO) .OR. (DABS(Y) .GE. XMIN)) GO TO 460
      K = K + 1
      GO TO 450
  460  MINEXP = -K
C------------------------------ --------------------------------
C      DETERMINE MAXEXP, XMAX
C---------------------------------------------------------------
      IF((MX.GT. K+K-3) .OR. (IBETA .EQ. 10)) GO TO 500
      MX = MX + MX
      IEXP = IEXP + 1
  500 MAXEXP = MX + MINEXP
C---------------------------------------------------------------
C      ADJUST FOR MACHINES WITH IMPLICIT LEADING
C     BIT IN BINARY SIGNIFICAND AND MACHINES WITH
C      RADIX POINT AT EXTREME RIGHT OF SIGNIFICAND
C---------------------------------------------------------------
      I = MAXEXP + MINEXP
      IF((IBETA .EQ. 2) .AND. (I .EQ. 0)) MAXEXP = MAXEXP - 1
      IF (I .GT. 20) MAXEXP = MAXEXP - 1
      IF(A.NE. Y) MAXEXP = MAXEXP -2
      XMAX = ONE - EPSNEG
      IF(XMAX*ONE .NE. XMAX) XMAX = ONE - BETA * EPSNEG
      XMAX = XMAX / (BETA * BETA * XMIN)
      I = MAXEXP + MINEXP + 3
      IF(I .LE. 0) GO TO 520
C
      DO 510 J = 1,I
      IF(IBETA .EQ.2)XMAX = XMAX + XMAX
      IF (IBETA .NE. 2) XMAX = XMAX * BETA
  510 CONTINUE
  520 RETURN
      END
```

APPENDIX B: LISTINGS OF PROGRAMS USED IN MATHEMATICAL TEST PROBLEMS

PROGRAM MAXJ

```
C     IMPLICIT DOUBLE PRECISION (A-H,O-Z)                                MAX00010
      SL = 0.                                                            MAX00020
      SU = 0.                                                            MAX00030
      READ (5,*)J                                                        MAX00040
      K = J + 1                                                          MAX00050
      DO 100 I = 1, J                                                    MAX00060
      VL = FLOAT(I)                                                      MAX00070
      VU = FLOAT(K - I)                                                  MAX00080
      SL = SL + VL                                                       MAX00090
      SU = SU + VU                                                       MAX00100
  100 CONTINUE                                                           MAX00110
      WRITE(6,10)SL,SU                                                   MAX00120
   10 FORMAT(' ',E14.8,2X,E14.8)                                         MAX00130
      DIFF = SU - SL                                                     MAX00140
      WRITE(6,20) DIFF                                                   MAX00150
   20 FORMAT(' ',' DIFFERENCE = ', F10.1)                                MAX00160
      STOP                                                               MAX00170
      END                                                                MAX00180
```

<u>Program MATRIX</u>

```
      REAL H(6,6),P,Q
      INTEGER I, J, K, N
      WRITE(6,800)
  800 FORMAT(1H1,40X,37HTHE SIX BY SIS MATRIX TO BE INVERTED //)
      READ(5,900)N
  900 FORMAT(I2)
C
C READ IN THE ORDER OF THE MATRIX AND COMPUTE THE MATRIX
C PRINT OUT THE ORIGINAL MATRIX BY ROWS AFTER COMPLETING THE INPUT PHASE
C
      DO 901 I = 1,N
      DO 901 J = 1,N
  901 H(I,J) = 1.0 / FLOAT(I+J-1)
C
      WRITE(6,801) ((H(I,J),J=1,N),I=1,N)
  801 FORMAT(1H ,10X,6E16.7//)
C
C     SELECT THE PIVOT P AND SET H(I,I) TO 1 IN THE OUTER LOOP
C
      DO 20 I = 1,N
      P = H(I,I)
      H(I,I) = 1.
C
C     DIVIDE ROW I BY THE SELECTED PIVOT P
C
      DO 30 J = 1,N
      H(I,J) = H(I,J) /P
   30 CONTINUE
C
C      MANIPULATE ALL ROWS OTHER THAN I TO EFFECT THE ZEROEING OF ALL
C      ELEMENTS IN THE MATRIX ABOVE AND BELOW THE SELECTED PIVOT
C
      DO 40 J = 1,N
         IF (I .EQ. J) GO TO 40
C
C        IF J NE J THEN SELECT THE ROW MULTIPLIERS Q AND ZERO H(I,J)
C
         Q = H(J,I)
          H(J,I) = 0.
C
C      MANIPULATE ALL COLUMS FOR ROW J SO THAT THE ELEMENT UNDER THE
C      PIVOT IS ZEROED
C
      DO 50 K = 1,N
      H(J,K) = H(J,K) - Q*H(I,K)
   50 CONTINUE
   40 CONTINUE
   20 CONTINUE
C
C THE ALGORITHM IS NOW COMPLETED AND THE INVERSE IS NOW IN THE ORIGINAL
C MATRIX H
C
C
      WRITE(6,801) ((H(I,J), J = 1,N), I=1,N)
      STOP
      END
```

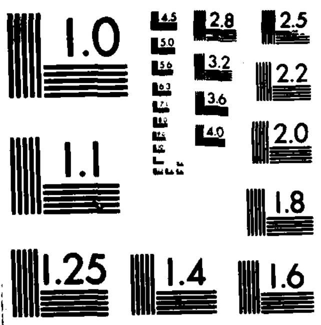
Program POLY

```
C NOISE LEVEL FOR POLYNOMIAL WHOSE ZEROS ARE FIRST N INTEGERS
C
C   PROGRAM EVALUATES POLYNONIAL FOR X = Z+J*H, J=-M(1)M.
C THEN COMPUTES MEAN AND STANDARD DEVIATION OF P-VALUES. CARE IS
C NEEDED TO CHOOSE H SMALL ENOUGH SO THAT TRUE P-VALUE ESSENTIALLY
C CONSTANT IN RANGE.
C
C 1. INPUT DEGREE OF POLYN. AND EVALUATE COEFFTS.
C
      DIMENSION A(20),P(21)
   10 WRITE (6,20)
   20 FORMAT(' ENTER N - DEGREE OF POLYN. TO STOP, ENTER N=0.')
C
      READ(5,*) N
      IF (N.EQ.0) STOP
      A(2) = -1
      A(1) = 1
      DO 40 K = 2,N
      A(K+1) = -K*A(K)
      DO 30 J = 1,K
      A(K-J+1) = A(K-J+1) - K*A(K-J)
   30 CONTINUE
   40 CONTINUE
      MMM = N + 1
      PRINT *,(A(I),I=1,MMM)
   50 WRITE (6,60)
   60 FORMAT ('ENTER Z,H,M FOR POLY. EVAL. M=0 TO ALTER POLY DEG')
      READ (5,*) Z,H,M
      IF  (M.EQ.0) GO TO 10
      MMM = 2*M+1
      DO 80 J = 1,MMM
      X = Z-(M-J+1)*H
      PVAL = A(1)
      DO 70 I =1,N
      PVAL = PVAL*X+A(I+1)
   70 CONTINUE
      P(J) = PVAL
   80 CONTINUE
      MMM = 2*M+1
C     PRINT *, (P(J),J=1,MMM)
      S = 0.0
      MMM = 2*M+1
      DO 90 K = 1,MMM
      S = S + P(K)
   90 CONTINUE
      PMEAN = S/(2*M+1)
      VAR = 0.0
      MMM = 2*M+1
      DO 100 I = 1,MMM
      VAR = VAR+(P(I) - PMEAN)**2
  100 CONTINUE
      RM = FLOAT(M)
      PSDEV = SQRT(VAR/(2.*RM))
      PRINT *,PMEAN,PSDEV
      GO TO 50
      END
```

AD-A134 141 ACCURACY CONSIDERATIONS WHEN USING SOME MINICOMPUTERS FOR SCIENTIFIC AND..(U) ARMY ENGINEER WATERWAYS EXPERIMENT STATION VICKSBURG MS W F INGRAM ET AL. 2/2

UNCLASSIFIED SEP 83 WES/TR/K-83-2 F/G 12/1 NL

END
DATE
FILMED
7 85


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

SUPPLEMENTARY

INFORMATION

AD-A134141

DEPARTMENT OF THE ARMY
WATERWAYS EXPERIMENT STATION, CORPS OF ENGINEERS
P.O. BOX 631
VICKSBURG, MISSISSIPPI 39180

REPLY TO
ATTENTION OF

WESKA 30 May 1985

Errata Sheet

No. 1

ACCURACY CONSIDERATIONS WHEN USING
SOME MINICOMPUTERS FOR SCIENTIFIC
AND ENGINEERING PROBLEMS

Technical Report K-83-2

September 1983

1. Pages 33, 34, and 36: Replace these pages with the inclosed corrected pages. (Double asterisks appear in the left margin to indicate those places that have been changed.)

HYDRAULICS LABORATORY GEOTECHNICAL LABORATORY STRUCTURES LABORATORY ENVIRONMENTAL LABORATORY COASTAL ENGINEERING RESEARCH CENTER

$$\begin{array}{c} \\ \\ \\ \\ ** \\ ** \end{array}\begin{bmatrix} 36 & -630 & 3360 & -7560 & 7560 & -2772 \\ -630 & 14700 & -88200 & 211680 & -220500 & 83160 \\ 3360 & -88200 & 564480 & -1411200 & 1512000 & -582120 \\ -7560 & 211680 & -1411200 & 3628800 & -3969000 & 1552320 \\ 7560 & -220500 & 1512000 & -3969000 & 4410000 & -1746360 \\ -2772 & 83160 & -582120 & 1552320 & -1746360 & 698544 \end{bmatrix}$$

Figure 2. Exact analytical solution of a 6 × 6 Hilbert matrix

$$\begin{bmatrix} \frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \\ \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} \end{bmatrix}$$

Figure 3. 6 × 6 Hilbert matrix

$$\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}$$

Figure 4. CDC approximate absolute error matrix
(* indicates value less than 1.0 E-6)

$$\begin{bmatrix} * & * & * & .1 & .1 & * \\ * & .1 & .7 & 1.7 & 1.9 & .7 \\ * & .7 & 4.4 & 11.6 & 12.8 & 5.1 \\ .1 & 1.7 & 11.5 & 30.1 & 33.3 & 13.2 \\ .1 & 1.9 & 12.8 & 33.3 & 36.8 & 64.5 \\ * & .7 & 5.0 & 13.1 & 64.5 & 5.7 \end{bmatrix}$$

a. Single- and double-precision

$$\begin{bmatrix} * & * & * & * & * & * \\ * & .1 & .3 & .9 & 1. & .4 \\ * & .3 & 2.3 & 6.1 & 6.7 & 2.6 \\ * & .9 & 6.1 & 15.8 & 17.4 & 6.9 \\ * & 1. & 6.7 & 17.4 & 19.3 & 57.6 \\ * & .4 & .3 & 6.9 & 57.6 & 3. \end{bmatrix}$$

b. Quadruple-precision

Figure 5. Harris approximate absolute error matrices (* indicates value less than 0.05)

$$\begin{bmatrix} 5.5 & 154.5 & 1034.5 & 2671.8 & 2933.6 & 1151.2 \\ 153.7 & 4311.3 & 28846.7 & 74455.0 & 81715.1 & 32057.2 \\ 1025.3 & 28744.3 & 192224.4 & 495963.9 & 54418.3 & 213443.6 \\ 2641.1 & 74000.4 & 494695.5 & 127607.4 & 139989.4 & 549005.2 \\ 2894.2 & 81058.2 & 341741.2 & 654081.0 & 153257.5 & 600934.0 \\ 1134.1 & 31751.1 & 311796.7 & 158732.5 & 269860.6 & 220955.7 \end{bmatrix}$$

a. Single-precision

$$\begin{array}{c} ** \\ ** \\ ** \\ ** \\ ** \\ ** \end{array} \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}$$

b. Double-precision

** **Figure 6. IBM approximate absolute error matrices (* indicates value less than 0.05)**

results have been rounded to 1 decimal place).

61. The CDC, the baseline system, has a very small absolute error matrix. The Harris is the only system with a small error matrix in single-precision. The double-precision approximate error matrices for the PRIME, IBM, and VAX were smaller than that for the Harris in quadruple-precision.

Real World Problems

62. For the S&E problems in this study, three computer programs were selected that have been in use for several years and have been thoroughly validated. These programs solve certain types of soil-structure interaction problems that are complex and for which no simple closed-form solution exist. Often the algorithm employed in the solution of a soil-structure interaction problem will yield incorrect results if the number of significant figures carried in the calculation is insufficient.

63. Three problems were chosen to show that different results can be obtained when identical problems are run on each of the systems tested using the same program. The problems have already been used in a previous study (O'Neil and Peterson 1976) with the same programs on a CDC system. The reported solutions from the CDC were duplicated in this study. The conclusions from the previous study indicated that the CDC's solutions were acceptable; therefore, these solutions are used for comparison with the results from the systems tested.

Case 1--flexible sheet pile bulkhead in sand

64. The deflections and moments produced in a flexible sheet pile bulkhead embedded in sand are to be computed for the physical system depicted in Figure 9. The problem was solved by using a computer code (BMCOL 28) which establishes the finite difference equations for a beam on an elastic subgrade at predesignated equally spaced nodes along the bulkhead (Matlock and Ingram 1963). The system of linear difference equations so generated forms a matrix equation in which the stiffness matrix is tightly banded. Recursive techniques are used to solve for the deflection at each node, and moments, shears, and soil reactions are subsequently computed.

65. It is common practice to analyze problems like that shown in Figure 9 by utilizing 50 to 100 nodes, but such solutions may be relatively

END

DATE

ILME